ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA DE LA SALUD

Sistema Web para la toma automatizada de constantes vitales en pacientes hospitalarios

Realizado por:

Elena María Espinosa García

Tutorizado por:

Tutorizado por María Mercedes Amor Pinilla

Departamento:

Lenguajes y ciencias de la computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, Junio 2019

Fecha defensa: Julio del 2019

El Secretario del Tribunal

Resumen

El objetivo de este proyecto es automatizar el tedioso trabajo de anotar los resultados de la toma de constantes a pacientes hospitalizados. De esta forma no sólo agilizamos el proceso, si no que también disminuimos la posibilidad del error humano al que se encuentra expuesto este trabajo rutinario y clave en la determinación de la evolución del paciente a lo largo de su estancia en el hospital.
Actualmente, esta rutina consiste en tomar las constantes a pacientes al menos tres veces al día y posteriormente anotar estas a mano en papel en lo que se denomina "gráfica del paciente", utilizando distintos colores para cada tipo de constante.
Es por ello que proponemos un sistema totalmente basado en tecnologías web que facilita este monótono trabajo, creando un flujo directo de información entre el dispositivo encargado de la toma de medidas y la historia clínica del paciente. Para ello, mediante una conexión Bluetooth Low Energy que se establece entre la aplicación web y el dispositivo medidor de constantes sin la necesidad de usar aplicaciones nativas, se automatiza una parte del proceso.
Esta aplicación también persigue el objetivo de que tanto médicos como el personal clínico encargado de la toma de medidas, estén informados en todo momento de la evolución clínica de sus pacientes. Para tal propósito se ha implementado una interfaz web que mediante su inicio de sesión permite al personal sanitario tener acceso a toda la información clínica relativa a la toma de constantes de sus pacientes, incluidas alarmas o advertencias sobre dichos pacientes que hayan sido incorporadas por compañeros de turnos anteriores.
Esta aplicación logra los siguientes objetivos fundamentales: minimizar del error humano, agilizar el proceso de la toma de constantes vitales, y facilitar la comunicación entre el personal sanitario, el cual incluye enfermeros, auxiliares de enfermería y médicos.

Esta aplicación ha sido implementada para tomar constantes como: temperatura, pulso

y frecuencia cardiaca. Sin embargo, su diseño, modular y extensible, sirve de base para el sistema se amplíe fácilmente para soportar la medición de otras las constantes vitales que son tomadas a pacientes hospitalizados.

Palabras clave: Bluetooth web api, Bluetooth low energy, Aplicación web, dispositivo central, dispositivo periférico, dispositivo medidor de constantes vitales.

Abstract

The purpose of this project is the automation of recollecting the hospitalized patients' vital signs. Usually, this work is done manually, which is a error prone task. In this way, we not only speed-up the process, but also we can reduce the human error. Now, the nurses should check the vital signs to patients at least three times a day, and write down, by hand, them in what is called "patient's graph", using different colours for each vital signs. In this work, we propose a fully web-based system that establises a direct flow between the device that takes the vital signs and the patient medical history, aoiding the manual annotation of any measurement. In order to achieve it, a Bluetooth low energy connection supports the communication between the health device and the nurse's device. An important benefit is that this connection does not need a native mobile app. Instead, the app runs ina mobile browser, and once the measure is taken by the health device and read by the personal device, it is sent to the patient medical history via http.

Also, a goal of this application is that nurses and doctors could be informed of their patients' evolution. To reach it, we have implemented a web app that allows the medical staff to have access to patient's evolution. It includes information about vital sign and alarms provided by nurses. With this functionality, the app gets the following objetives: Minimize the human error, speed up the process of taking vital signs, and a common communication platform between the health staff members.

This application have been implemented only for the following vital constants: temperature, blood pressure and heart rate; but its modular and extensible design serves as the basis to cover the rest of vital signs taken to hospitalized patients.

Key words: Bluetooth web api, Bluetooth low energy, web application, beacon, central device, peripheral device.

Contents

1 Introducción **13**

1.1 Contexto 13

1.2 Motivación 14

1.3 Resultados esperados 15

1.4 Estructura del documento 16

2 Objetivos del proyecto **19**

3 Estado del arte **21**

4 Métodos **25**

4.1 Tecnologías Utilizadas 25

4.1.1 Python 25

4.1.2 Node.js 26

4.1.3 React 26

4.1.4 Web Bluetooth API 27

4.1.5 Dispositivo bluetooth 28

4.1.6 Sqlite3 28

4.1.7 Css, React-Bootstrap y html5 28

4.2 Metodología 29

4.3 Línea temporal 31

5 Análisis del sistema **33**

5.1 Requisitos funcionales . . . 33

5.2 Requisitos no funcionales . . . 34

5.3 Casos de uso . . . 35

5.3.1 Diagrama de casos de uso . . . 36

5.3.2 Descripción casos de uso . . . 37

6 Diseño **47**

7 Implementación **55**

7.1 Conexión con el dispositivo periférico . . . 55

7.1.1 Componente Bluetooth . . . 55

7.1.2 Componente HeartRate . . . 57

7.1.3 Componente BloodPresure . . . 59

7.1.4 Componente Temperature . . . 60

7.2 Gestión de pacientes . . . 61

7.2.1 Componente Main_Nurse . . . 61

7.2.2 Componente Planner . . . 62

7.2.3 Componente FormAlert . . . 63

7.2.4 Componente List_patients . . . 64

7.2.5 Componentes NurseHistory y MedicineHistory . . . 65

7.3 Autenticación del personal sanitario y del paciente . . . 66

7.4 Cambio de contraseña . . . 69

7.5 Renderización de React en el Servidor . . . 70

8 Evaluación del impacto **73**

9 Líneas futuras **75**

10 Conclusiones finales **77**

Apéndice A **81**

A Implementación **81**

A.1 Componente Bluetooth 82
A.2 Componente Temperature 86
A.3 Componente BloodPresure 90
A.4 Componente HeartRate 94
A.5 Modelo 98
A.6 Vista 102

Apéndice B **114**

B Manual de instalación **115**

B.1 Requisitos previos 116
B.2 Conocimientos previos 116
B.3 Ejecución del proyecto 116
B.4 Aspectos a considerar 118

Chapter 1

Introducción

En este capítulo se va a proporcionar al lector una visión general sobre el proyecto desarrollado. Para ello se ha dividido en los siguientes apartados: contexto, motivación, objetivos establecidos y estructura de la memoria.

1.1 Contexto

Actualmente vivimos en un mundo que aunque está cada vez más informatizado sigue utilizando el papel y el trabajo manual como método de recolección de datos. Sin embargo, la importancia de disponer de estos datos en cualquier momento y lugar, y la necesidad de optimizar cada vez más los procesos con la finalidad de minimizar el error y mejorar el rendimiento, lleva a la urgencia de la automatización de los mismos.

En el contexto de un hospital, el papel juega un papel protagonista en la recopilación de datos clínicos. Esto no sólo ralentiza el trabajo del personal sanitario, si no que también dificulta que este pueda llevar un seguimiento continuo y cercano del paciente. Es por ello que, cada vez surgen nuevas tecnologías que pretenden hacer frente a esto. Un ejemplo de aplicación de nuevas tecnologías en un contexto clínico para la recolección de datos

es, por ejemplo Hydra Analytics[1], el primer enfermero virtual probado con éxito por la Mutua Montañesa, que permite obtener datos biométricos en tiempo real de pacientes y enviarlos a profesionales en cualquier parte del mundo.

Este proyecto nace con la finalidad de automatizar un proceso rutinario y monótono como es la toma de las constantes vitales y la anotación digital de las mismas en la gráficas digitales de pacientes hospitalizados. De esta forma, no solo facilitamos y aceleramos el trabajo al personal sanitario, también permitimos que se pueda llevar el seguimiento telemático del paciente desde cualquier parte.

1.2 Motivación

Aunque el concepto de automatización puede asustar un poco, este es un sinónimo de progreso. ¿Se imagina un mundo en el que el papel sólo forme parte del pasado? Donde toda anotación se realice de forma digital y automática, logrando así una optimización del trabajo del personal sanitario y permitiendo el acceso a los datos de forma instantánea y desde cualquier punto.

Con esta aplicación pretendemos automatizar un proceso rutinario y monótono como es el de la anotación de valores de constantes vitales de pacientes hospitalizados mediante una aplicación web que establece conexión bluetooth low energy con un beacon encargado de tomar dichas medidas. De esta forma no solo simplificamos el trabajo del personal sanitario a labores meramente clínicas, también logramos una mejora en la comunicación multidireccional entre médicos, enfermeros y pacientes y un acceso inmediato a información vital del paciente.

Todo esto tiene un gran impacto en la calidad de la atención y el cuidado que recibe el paciente, pues además de permitir al personal sanitario llevar a cabo un seguimiento continuo y de cerca de el, también protegemos la integridad de sus datos, que puede verse dañada debido a la pérdida de información asociada al error humano y de un posible acceso no autorizado a información confidencial y sensible que pueda violar su privacidad.

Además el uso de tecnologías inalámbricas, como bluetooth, no sólo aumenta la precisión en la medida, también permite realizar mediciones menos intrusivas y reduce la infraestructura necesaria para llevar a cabo tal medición. Esto último reduce el equipamiento necesario a un ipad o cualquier otro dispositivo móvil con conexión bluetooth y conexión a la red y el dispositivo, con conunicación Bluetooth, requerido para tal medición.

1.3 Resultados esperados

Los resultados que cabe esperar de este trabajo son dos aplicaciones web, una orientada a enfermeros y auxiliares de enfermería y otra a médicos. La primera de ellas será la encargada de la adquisición de constantes vitales del paciente y de la vinculación e incorporación de las lecturas al historial médico digital del paciente. Esta también ofrecerá la posibilidad de crear alertas relativas a pacientes con la finalidad de que estas sean tenidas en cuenta por los compañeros del siguiente turno y la visualización de la evolución del paciente. La otra aplicación, sólo tendrá la funcionalidad de lectura y les proporcionará a los facultativos la posibilidad de observar información vital de sus pacientes a lo largo de su trayectoria en el hospital.

1.4 Estructura del documento

A continuación daremos una pequeña pincelada sobre el contenido de los siguientes capítulos:

- Capítulo 2:

  En el capítulo 2 describiremos los objetivos de nuestro proyecto, tanto generales, como específicos.

- Capítulo 3:

  En el capítulo 3 describiremos el estado del arte, donde realizaremos una revisión de la situación actual de esta área de trabajo y explicaremos que aporta nuestra aplicación a esta área.

- Capítulo 4

  El capítulo cuatro se compone de tres secciones:

  1. Tecnologías empleadas
  2. Metodología empleada para el desarrollo del proyecto
  3. Línea temporal, es decir, tiempo estimado para la realización de cada una de las tareas propuestas en la sección anterior.

- Capítulo 5

  En el capítulo 5 realizaremos un análisis del sistema que escalarecerá el alcance de este. Describiremos los requisitos funcionales y no funcionales de nuestra aplicación así como los casos de uso.

- Capítulo 6

  El capítulo 6 mostrará el diseño de aplicación. Aquí describiremos la arquitectura general de la aplicación.

- Capítulo 7

  Este capítulo muestra la implementación, es decir el software que ha hecho posible el diseño de la aplicación. Aquí describiremos muy brevemente los componentes más esenciales, aportando en algunos casos breves fragmentos de código.

- Capítulo 8

  En este capítulo realizaremos un análisis del impacto que creeemos que ha podido tener nuestra aplicación en el sector sanitario, concretamente en el área de urgencias.

- Capítulo 9

  El capítulo nueve describirá las limitaciones de nuestro proyecto, proponiendo así nuevas ideas para seguir avanzando en este área.

- Capítulo 10

  Finalmente, el proyecto termina con las conclusiones finales donde realizaremos una síntesis de nuestro proyecto.

Chapter 2

Objetivos del proyecto

El propósito de este proyecto es lograr una automatización del proceso de anotación de constantes vitales a pacientes hospitalizados mediante el establecimiento de un flujo de información directo entre el dispositivo Bluetooth encargado de la medición y el historial médico del paciente. De esta forma pretendemos lograr una optimización del trabajo del personal sanitario y una disponibilidad inmediata de estos datos.

Todo esto permite así mismo:

1. la eliminación del error en la anotación de las constantes del paciente.
2. Dejar constancia del estado del paciente en cada momento.
3. Minimizar el tiempo invertido en el proceso.
4. Mejora de la comunicación multidireccional entre el personal sanitario.
5. Permitir un seguimiento del paciente desde cualquier punto.
6. reducir la intrusión al paciente en la medición
7. salvaguardar la privacidad del paciente

8. lograr mayor precisión en la medición

Chapter 3

Estado del arte

El proceso de anotación de constantes a pacientes hospitalizados o en quirófano es una labor que solía y aun en algunos hospitales suele hacerse a mano.

Este trabajo resulta muy tedioso, especialmente en quirófano, donde los anestesistas deben anotar a mano y cada cinco minutos los valores reflejados en el monitor o en UCI, donde los enfermeros deben tomar y anotar los valores de temperatura cada quince minutos.

Esta labor además esta sujeta a error, lo que puede traducirse en la muerte del paciente cuando hablamos de pacientes en UCI donde cada acción por pequeña que sea es crucial y determinante.

En el caso de pacientes hospitalizados durante un largo periodo, esto también imposibilita al facultativo la disponibilidad inmediata y desde cualquier punto de la evolución de su paciente, dificultando así el seguimiento continuo y de cerca del mismo. Además esto puede suponer una violación de la privacidad del paciente, pues al encontrarse estos datos anotados a papel, personas no autorizadas podrían tener acceso a información sensible del paciente, por lo que llevar un control de quien y cuando se accede a los datos

resulta fundamental.

Por todo esto, a lo largo de estos años numerosas empresas han centrado su trabajo en el diseño de nuevas tecnologías que permitan automatizar este trabajo.

Un ejemplo de ello es HydraAnalytics[1], el primer enfermero virtual, diseñado por Mensoff y probado por éxito en mutua Montañesa, que permite la obtención de constantes vitales de pacientes en tiempo real para su envío a profesionales de todo el mundo. Esto se realiza mediante un conjunto de sensores que permiten la monitorización del paciente. De esta forma si hubiese una urgencia médica y ningún personal cualificado cerca, mediante HydraAnalytics se podría realizar una evaluación inicial del paciente.

Otro ejemplo de avance en este área sería "B anesthesic" diseñado para ser utilizado en quirófano. Este permite la anotación automática de las constantes vitales del paciente. Su diseño permite la identificación automática del paciente mediante una pulsera que a su vez se conecta al monitor al que también está conectada la aplicación que realiza la anotación automática.

Estas tecnologías suponen un gran avance en el ámbito hospitalario, sin embargo dejan desprotegida al área de hospitalización, donde también se requiere una automatización de la anotación de constantes vitales a pacientes.
Es por ello que pensamos en diseñar una tecnología que hiciese esta anotación de forma automática. Numerosos estudios anteriores habían reflejado la posibilidad de establecer conexión entre dispositivos Bluetooth y sitios web[2][3][4] y sobre la eficacia y rendimiento de bluetooth Low Energy (BLE)[5][6]. Esto nos llevó al diseño de una aplicación web que establece conexión BLE [7] [8] [9]. Las aplicaciones HealthManager y BabyCare, implementadas por la empresa alemana Beurer, también sentaron las bases de este proyecto.

Estas aplicaciones establecen conexión mediante aplicaciones nativas y están destinadas al uso personal por lo que decidimos diferenciarnos de ellas realizando esta conexión desde un sitio web y llevándola al ámbito hospitalario.

Esta aplicación no cubrirá la automatización de la anotación de todas las constantes vitales pero pretende establecer un punto de inicio para seguir avanzando en esta área.

Chapter 4

Métodos

4.1 Tecnologías Utilizadas

En esta sección se procede a describir las tecnologías utilizadas a lo largo del desarrollo del proyecto así como el impulso que nos ha llevado a usar estas y no otras.

4.1.1 Python

Python[10] ha jugado un papel fundamental en nuestro proyecto, pues para la implementación de la api que alimenta nuestra aplicación hemos utilizado el framework Django. Django[11][12][13][14][15] es un framework de alto nivel escrito en Python y por tanto hereda todas las características y facilidades de este, permitiéndonos así construir una aplicación estable, potente y segura de forma rápida. Podemos destacar algunas de las características principales de este framework que lo convierten en la herramienta estrella para la construcción de aplicaciones web.

- **Don't repeat yourself (DRY):** el diseño de django permite la reutilización de código.
- **Admin :** Django trae consigo un sistema de administración activo que no requiere

ningún tipo de configuración.

- **ORM (object-relational mapping) :** esta implementación, entre otras cosas, nos permite interaccionar con la base de datos sin necesidad de utilizar el lenguaje SQL.

4.1.2 Node.js

Node.js es un entorno de ejecución para JavaScript utilizado también para la construcción del backend. El propósito de añadir también esta tecnología a nuestro proyecto tiene su razón de ser en react. React, librería de javascript de la cual hablaremos a continuación, permite la renderización en el servidor(SSR) mediante el uso de node.js consiguiendo así:

1. Aumentar la velocidad de la carga inicial.
2. Mejorar el SEO al mejorar legibilidad a los buscadores aumentando así la velocidad de aparición y clasificación en los resultados de búsqueda.

Así, cuando el navegador solicita el sitio web a Django, Django desviará la petición a node.js, que renderizará los componentes de react y devolverá al cliente el contenido ya cargado.

4.1.3 React

React [16], tal y como se ha mencionado en la sección anterior, es una librería de javascript cuya finalidad es la de crear interfaces de usuario. Esta librería realiza una fusión de html y javascript en un sript jsx y se basa en el uso de componentes que envuelve tanto la lógica como la presentación.

A continuación mencionaremos unas de las ventajas principales de usar esta librería:

- **DOM virtual:** el DOM se genera de forma automática, haciendo innecesario renderizar la página de nuevo al completo si hay cambio en los datos.
- **Babel:** gracias a esta herramienta, el código JavaScript escrito con la especificación ECMAScript 6 es transformado para que pueda ser interpretado por cualquier navegador.
- **Reutilización de los componentes** convirtiéndola en una aplicación escalable y fácil de mantener.
- **Isomorfismo** o **Javascript Universal**: capacidad de ejecutar código tanto en el lado del servidor como en el cliente.
- **Desarrollo declarativo** gracias a los estados.

4.1.4 Web Bluetooth API

Esta api[4] es la protagonista de nuestra aplicación, pues es la que nos ha permitido conectar el dispositivo central al periférico mediante una conexión BLE (bluetooth low energy). Denominamos dispositivo central o cliente GATT, al dispositivo desde el cual se accede a la aplicación y periférico o servidor GATT al dispositivo encargado de la toma de constantes al paciente. De esta forma, gracias a esta api, cliente y servidor pueden intercambiar información de forma bidireccional utilizando servicios y características.

Hasta ahora, este tipo de aplicaciones se habían implementado mediante el desarrollo de aplicaciones nativas, por lo que la razón de usar esta api y no la Bluetooth api para Android (o ios) tiene su incentivo en la motivación de desarrollar algo innovador y multiplataforma, pues como dijo Albert Einstein: "si buscas resultados distintos, no hagas siempre lo mismo".

4.1.5 Dispositivo bluetooth

Este sería el servidor GATT y dispositivo periférico. Para la validación del proyecto hemos utilizado dispositivos bluetooth virtuales gracias a la aplicación: LightBlue, que nos ha permitido simular un termómetro, un tensiómetro y un pulsiómetro o electrocardiograma.

4.1.6 Sqlite3

Sqlite3 será nuestro gestor de base de datos. La elección de este, el cual viene configurado por defecto en Django se debe a que este gestor es más eficiente y más rápido que otros gestores de base de datos sql como MySQL o PostgreSQL. El inconveniente de este gestor es que está diseñado para base de datos de pequeña magnitud pero al ser esta una pequeña representación de la realidad, pensamos que la elección de este para nuestro proyecto sería lo más correcto.

Por otro lado, la elección de una base de datos sql y no una xml o noSql se sebe a que gracias a su madurez y sencillez en la escritura, podemos intercambiar datos de forma sencilla y rápida.

4.1.7 Css, React-Bootstrap y html5

Estas herramientas son las que nos han ayudado a convertir nuestra aplicación en una aplicación atractiva e intuitiva. React-Bootstrap y html5 (HyperText Markup Language, versión 5) nos han permitido construir los elementos que forman los componentes de nuestra aplicación. Css(Cascading Style Sheets), por otro lado nos ha permitido estilizar estos componentes.

4.2 Metodología

La metodología utilizada en este proyecto se corresponde con una metodología ágil, apoyándose en el propósito de la metodología SCRUM4.1. El funcionamiento de esta se basa en dividir el proyecto final en subproyectos, cada uno dividido así mismo en tres etapas: análisis, desarrollo y testing. La finalidad de este método ágil es ir realizando pequeñas entregas funcionales.

Así, aplicando esta metodología a nuestro proyecto, las distintas subetapas de la etapa 3 que se muestran a continuación se realizarán en orden y se irán convirtiendo en pequeñas entregas increméntales de modo que en cada entrega o sprint se deberá lograr la funcionalidad de esa etapa unida a las etapas anteriores, logrando así entregas regulares y parciales cuya suma es la funcionalidad completa del proyecto final.

A continuación se procede a enumerar las distintas etapas correspondientes al desarrollo del software iterativo.

1. Implementación de una API REST que permita mostrar, agregar, actualizar o borrar datos relativos a pacientes o al personal sanitario. No obstante esta etapa se irá desarrollando en las siguientes etapas aunque no se mencione explícitamente.

2. Implementación de una aplicación que permita identificar al paciente mediante la lectura de un código de barras y al enfermero mediante la aportación de un código identificativo y una contraseña.

3. Agregar la opción de cambiar contraseña a enfermeros o auxiliares de enfermería.

4. Aumentar la funcionalidad, haciendo que esta sea capaz de establecer conexión con el dispositivo bluetooth y de enviar la información recogida a la base de datos.

5. incrementar la funcionalidad de la etapa anterior permitiendo al enfermero o auxiliar de enfermería crear alertas, visualizar la evolución del paciente y consultar su agenda de pacientes.

6. Implementación de una aplicación que permita el inicio de sesión del facultativo mediante un código identificativo y una contraseña.

7. Agregar la opción de cambiar contraseña al facultativo.

8. Aumentar la funcionalidad de la etapa anterior de forma que permita al doctor visualizar información funcional de cada uno de sus pacientes.

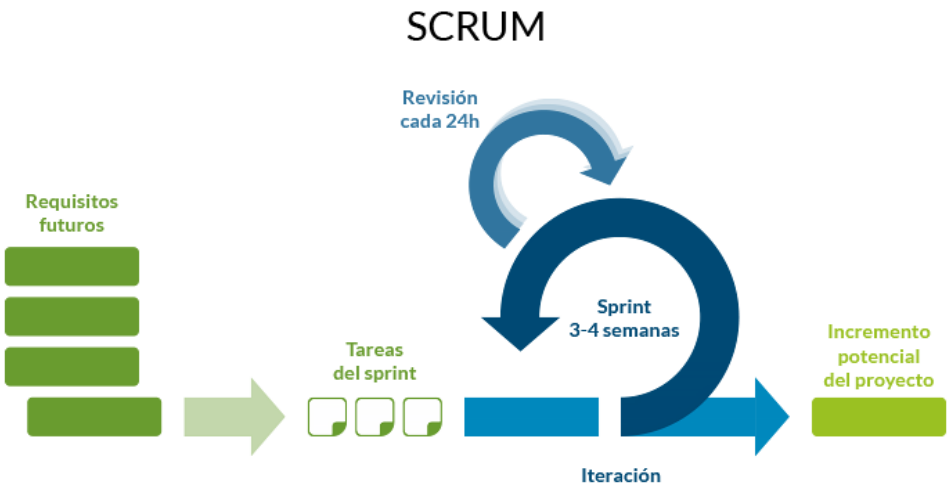

Figure 4.1: Metodología scrum

4.3 Línea temporal

Por último mostraremos las horas estimadas para la realización de cada una de las tareas anteriormente propuestas:

1. **Documentación:** 60 horas
2. **Instalación, configuración y despliegue de la base de datos:** 50 horas
3. **Desarrollo de Software Iterativo:** 141 horas
4. **Creación del manual de usuario y redacción de la memoria:** 45 horas

Chapter 5

Análisis del sistema

En este punto estableceremos cual será el comportamiento de nuestro sistema, es decir, los servicios que proporcionará así como sus propiedades.

Este punto es muy importante pues nos permite definir cual será el alcance de nuestro proyecto.

5.1 Requisitos funcionales

Los requisitos funcionales como su palabra indica son las funciones que tendrá el sistema, es decir, los servicios que prestará.

A continuación procederemos a describir cada uno de ellos.

Gestión de usuarios:

- **Requisito funcional 01 :** el sistema permitirá al personal sanitario iniciar sesión en la aplicación mediante la aportación de un usuario y contraseña.

- **Requisito funcional 02 :** el sistema permitirá al personal sanitario cambiar su contraseña en la aplicación.
- **Requisito funcional 03 :** el sistema permitirá la identificación del paciente mediante la lectura de un código de barras.

Funciones del sistema:

- **Requisito funcional 04 :** el sistema realizará la anotación de constantes vitales de forma automática al historial médico del paciente.
- **Requisito funcional 05 :** el sistema permitirá al usuario auxiliar o enfermero visualizar su agenda de pacientes.
- **Requisito funcional 06 :** el sistema permitirá al usuario auxiliar o enfermero crear alarmas relativas a pacientes.
- **Requisito funcional 07 :** el sistema permitirá al usuario auxiliar o enfermero visualizar la evolución de sus pacientes.
- **Requisito funcional 08 :** el sistema permitirá al usuario facultativo observar información de constantes relativas a sus pacientes.

5.2 Requisitos no funcionales

Los requisitos no funcionales hacen referencia a las propiedades emergentes de nuestro sistema describiendo así la calidad del mismo.

A continuación procedemos también a la enumeración de los mismos:

- **Requisito no funcional 01 :** la aplicación podrá ser ejecutada en dispositivo con conexión a internet y conexión bluetooth.

- **Requisito no funcional 02 :** la aplicación solo es soportada por Chrome y Opera a partir de las versiones que se muestran a continuación:
  - **Chrome:** versión 56 o superior.
  - **Chrome para Android:** versión 56 o superior.
  - **Opera:** versión 43 o superior.
  - **Opera para Android:** versión 43 o superior.
- **Requisito no funcional 03 :** la aplicación es soportada por la mayoría de las plataformas a partir de las siguientes versiones:
  - **iOS:** iOS5+, preferentemente iOS7+.
  - **Android:** Android 4.3+.
  - **Windows:** Windows 8.
  - **Linux:** Linux Vanilla BlueZ 4.93+.
- **Requisito no funcional 04 :** la aplicación debe estar a un máximo de 100 m del beacon para garantizar su buen funcionamiento.

5.3 Casos de uso

En este apartado procedemos a describir los casos de uso, es decir, la secuencia de acciones que llevará a cabo el sistema visto desde el punto de vista del usuario, viendo así al sistema como una historia la cual tiene unas acciones, y un contexto.
En un primer lugar mostraremos el diagrama de casos de uso y en segundo lugar una descripción detallada de los casos de uso.

5.3.1 Diagrama de casos de uso

El diagrama de casos de uso pretende mostrar las interacciones entre sistema y usuario de una forma visual mediante una representación UML.

Este queda mostrado a continuación:

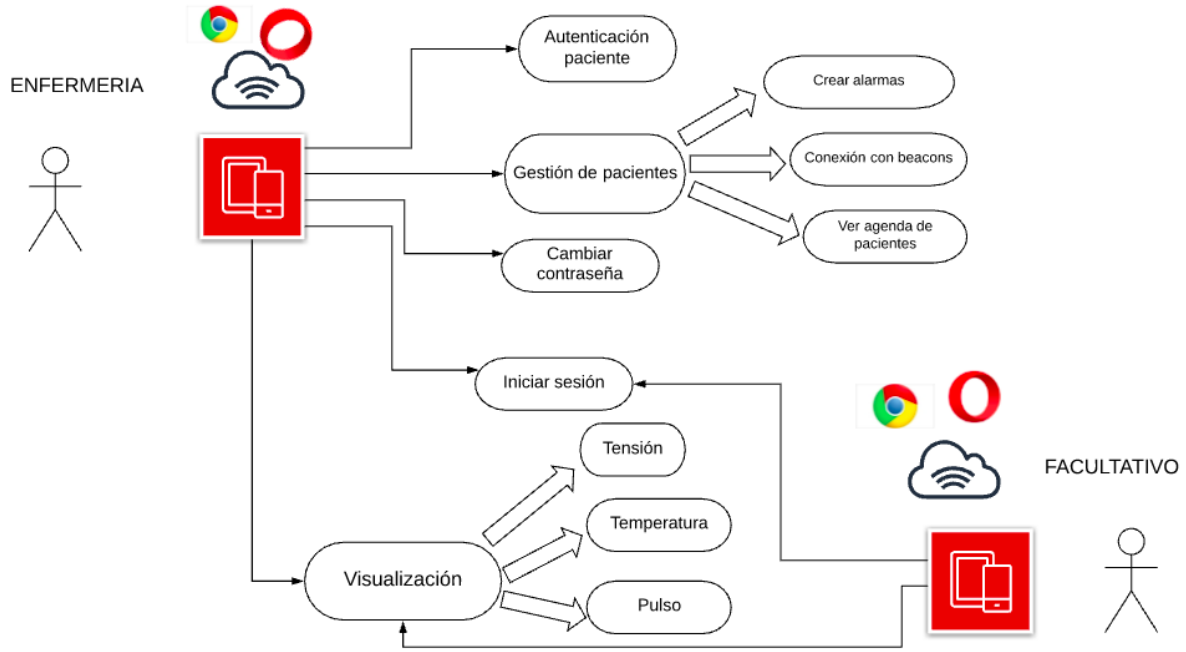

Figure 5.1: Diagrama de casos de uso

5.3.2 Descripción casos de uso

- **Caso de uso 01 :** el usuario enfermero o auxiliar de enfermería podrá iniciar sesión en la aplicación mediante la aportación de un usuario y contraseña. Tras el inicio de sesión entrará en la pantalla principal donde podrá:
  - Crear alarmas relativas a pacientes.
  - Identificación del paciente.
  - Consulta agenda de pacientes.
  - Visualizar evolución del paciente.

  - **Usuario principal :** auxiliar de enfermería y enfermero.
  - **Ámbito :** aplicación web.
  - **Nivel :** nivel de usuario.
  - **Participantes e intereses :**
    - enfermería: entrar al sistema para la gestión de pacientes.
  - **Precondición:** el usuario entra correctamente en la aplicación.
  - **Garantía mínima:** el usuario sabrá si ha introducido usuario y contraseña correctos.
  - **Garantía de éxito:** el usuario accede a la pantalla principal de la aplicación donde se le mostrarán todas las funcionalidades de esta.
  - **Escenario principal:**
    1. el usuario accederá a la aplicación.
    2. el usuario introducirá usuario y contraseña.
    3. el sistema validará usuario y contraseña.

4. el sistema redirige al usuario a la pantalla principal que consta de las siguientes funcionalidades:
   * Crear alarmas relativas a pacientes.
   * Identificación del paciente.
   * Consulta agenda de pacientes.
   * Visualizar evolución del paciente.

– **Escenario alternativo:**

2a. el usuario introduce usuario y contraseña incorrectas.

2a1. el sistema muestra un mensaje de error.

- **Caso de uso 02 :** el usuario facultativo podrá iniciar sesión en la aplicación mediante la aportación de un usuario y contraseña. Tras el inicio de sesión entrará en la pantalla principal donde podrá ver información de constantes de sus pacientes.

  – **Usuario principal :** facultativo.

  – **Ámbito :** aplicación web.

  – **Nivel :** nivel de usuario.

  – **Participantes e intereses :**

    * facultativos: entrar al sistema para visualizar la evolución de sus pacientes.

  – **Precondición:** el usuario entra correctamente en la aplicación.

  – **Garantía mínima:** el usuario sabrá si ha introducido usuario y contraseña correctos.

  – **Garantía de éxito:** el usuario accede a la pantalla principal de la aplicación donde se le mostrarán todas las funcionalidades de esta.

  - **Escenario principal:**
    1. el usuario accederá a la aplicación.
    2. el usuario introducirá usuario y contraseña.
    3. el sistema validará usuario y contraseña.
    4. el sistema redirige al usuario a la pantalla principal donde podrá visualizar la evolución del paciente seleccionado.
  - **Escenario alternativo:**

    2a. el usuario introduce usuario y contraseña incorrectas.

    2a1. el sistema muestra un mensaje de error.

- **Caso de uso 03:** el usuario podrá cambiar la contraseña.
  - **Usuario principal:** personal sanitario.
  - **Ámbito:** aplicación web.
  - **Nivel:** nivel de usuario.
  - **Participantes e intereses :**
    * facultativos: cambiar contraseña aplicación.
    * enfermería: cambiar contraseña aplicación.
  - **Precondición:** el usuario entra correctamente en la aplicación.
  - **Garantía mínima:** el usuario sabrá si ha cambiado la contraseña correctamente.
  - **Garantía de éxito:** el usuario podrá cambiar su contraseña.

- **Escenario principal:**
  1. el usuario accederá a la aplicación.
  2. el usuario seleccionará la opción: cambiar contraseña.
  3. el sistema pedirá al usuario que ingrese usuario, contraseña antigua, nueva contraseña y tipo de usuario.
  4. el sistema lanzará un mensaje de verificación.
  5. el sistema redirigirá al usuario a la pantalla de inicio.
- **Escenario alternativo:**

  4a. el sistema lanza un mensaje de error.

- **Caso de uso 04 :** el usuario enfermero o auxiliar de enfermería podrá realizar la autenticación al paciente e iniciar la anotación automática de constantes.
  - **Usuario principal:** auxiliar de enfermería o enfermero.
  - **Ámbito:** aplicación web.
  - **Nivel:** nivel de usuario.
  - **Participantes e intereses :**
    * auxiliar de enfermería o enfermero: iniciar la anotación automática de valores de constantes vitales al historial médico.
  - **Precondición:** el usuario pulsa la opción: tomar medida.
  - **Garantía mínima:** el usuario sabrá si ha realizado correctamente la autenticación del paciente.
  - **Garantía de éxito:** el usuario podrá iniciar la anotación automática.

- **Escenario principal:**

  1. el usuario selecciona la opción tomar medida.
  2. el usuario autentifica al paciente en la aplicación.
  3. el sistema validara al paciente.
  4. el sistema habilita la opción: comenzar medición que tal ser pulsada inicia automáticamente la anotación de constantes.
  5. el sistema pide al usuario que seleccione el dispositivo al que desea conectarse.
  6. el sistema comienza la anotación automática.
  7. el sistema muestra el resultado de la medición y las opciones: enviar y repetir medición.
  8. el usuario selecciona la opción: enviar.
  9. el sistema realiza la anotación automática.
  10. el sistema redirige al usuario a la pantalla principal.

- **Escenario alternativo:**

  2a. el usuario no pulsa correctamente la opción: tomar medida.

  2a1. el usuario sigue en el menú principal.

  4a. el paciente no es reconocido por el sistema.

  4a1. el sistema muestra un mensaje de error.

  4a2. el sistema redirige al usuario al menú principal.

  6a. el sistema no interacciona correctamente con el dispositivo bluetooth.

  6a1. el sistema lanza un mensaje de error.

  6a2. el sistema redirige al usuario al menú principal

  9a. el usuario selecciona la opción: repetir.

  9a1. el sistema volverá a leer el resultado de la medición.

9a2. el sistema muestra el resultado de la medición y las opciones: enviar y repetir medición.

- **Caso de uso 05 :** el usuario enfermero o auxiliar de enfermería podrá consultar su agenda de pacientes.
  - **Usuario principal:** auxiliar de enfermería o enfermero.
  - **Ámbito:** aplicación web.
  - **Nivel:** nivel de usuario.
  - **Participantes e intereses :**
    - auxiliar de enfermería o enfermero: consultar su agenda de pacientes de hoy.
  - **Precondición:** el usuario pulsa la opción consultar agenda.
  - **Garantía mínima:** el usuario accederá a una nueva ventana.
  - **Garantía de éxito:** el usuario accederá a una nueva ventana donde se le mostrarán sus pacientes y donde podrá comprobar los ya visitados .
  - **Escenario principal:**
    1. el usuario selecciona la opción consultar agenda.
    2. el usuario podrá consultar y gestionar su agenda.
  - **Escenario alternativo:**

    2a. el usuario no pulsa correctamente la opción: consultar agenda.

    2a1. el usuario sigue en el menú principal.

- **Caso de uso 06 :** el usuario enfermero o auxiliar de enfermería podrá crear alertas relativas a pacientes.

- **Usuario principal:** auxiliar de enfermería o enfermero.
- **Ámbito:** aplicación web.
- **Nivel:** nivel de usuario.
- **Participantes e intereses :**
  - auxiliar de enfermería o enfermero: crear alertas relativas a pacientes.
- **Precondición:** el usuario pulsa la opción: crear alerta
- **Garantía mínima:** el usuario accederá a una nueva ventana donde se le mostrará un formulario.
- **Garantía de éxito:** el usuario enviará la alerta correctamente.
- **Escenario principal:**
  1. el usuario selecciona la opción crear alerta.
  2. el usuario podrá rellenar un formulario con la alerta.
  3. el usuario podrá seleccionar la opción: enviar la alerta.
  4. el sistema enviará la alerta.
- **Escenario alternativo:**

  2a. el usuario no pulsa correctamente la opción: crear alerta.

  2a1. el usuario sigue en el menú principal.

  4a. la alerta no se envía correctamente.

  4a1. el sistema lanza un mensaje de error.

  4a2. el sistema redirige al usuario al menú principal.

- **Caso de uso 07 :** el usuario enfermero o auxiliar de enfermería podrá visualizar la evolución de sus pacientes.

- **Usuario principal:** auxiliar de enfermería o enfermero.
- **Ámbito:** aplicación web.
- **Nivel:** nivel de usuario.
- **Participantes e intereses:**
  - auxiliar de enfermería o enfermero: visualizar evolución de sus pacientes.
- **Precondición:** el usuario pulsa la opción: ver pacientes.
- **Garantía mínima:** el usuario accederá a una nueva ventana.
- **Garantía de éxito:** el usuario podrá visualizar valores de constantes y alarmas del paciente seleccionado.
- **Escenario principal:**
  1. el usuario selecciona la opción ver pacientes.
  2. el usuario selecciona el paciente a visualizar.
  3. el usuario podrá ver la evolución del paciente.

     3.1 el usuario podrá ver información de constantes.

     3.1 el usuario podrá ver alarmas.

     3.1 el usuario podrá borrar alarmas.
- **Escenario alternativo:**

  1a. El usuario no pulsa correctamente la opción: ver pacientes.

  1a1. el usuario sigue en el menú principal.

  3a. la alerta no se borra correctamente.

  3a1. el sistema lanza un mensaje de error.

- **Caso de uso 08 :** el usuario facultativo podrá visualizar la evolución de sus pacientes.

- **Usuario principal:** facultativo.
- **Ámbito:** aplicación web.
- **Nivel:** nivel de usuario.
- **Participantes e intereses :**
  - médico: visualizar evolución de su paciente.
- **Precondición:** el usuario inicia sesión correctamente.
- **Garantía mínima:** el usuario accederá a una nueva ventana.
- **Garantía de éxito:** el usuario podrá visualizar el historial de constantes del paciente seleccionado.
- **Escenario principal:**
  1. el usuario entra en la pantalla principal.
  2. el usuario selecciona el usuario a visualizar.
  3. el usuario podrá ver el historial de valores de constantes del paciente.
- **Escenario alternativo:**

  1a. el sistema no introduce usuario y contraseña correctos.

  1a1. el sistema muestra un mensaje de error.

Chapter 6

Diseño

Este proyecto (tal y como muestra la figura 6.6) implementa dos aplicaciones web destinadas al ámbito clínico. Cada una de ellas está destinada a un sector distinto, y por tanto funcionan de forma independiente y están destinados a dos perfiles de usuarios distintos. Una de ellas está destinada a la enfermería (personal clínico) y la otra a los facultativos. Los *front-ends* de ambas están diseñados con React y comparten un mismo *back-end*, implementado en Django, que actúa como API REST. El *front-end* de ambas es renderizado en el servidor en vez de en el navegador. De esta forma, tal y como se indicó en los objetivos, conseguimos:

1. Rapidez en la carga inicial.

2. Mejorar el posicionamiento del SEO.

Para llevar a cabo la renderización en el servidor(SSR) utilizamos node.js, un entorno de ejecución para javascript, y realizamos una comunicación entre django y node.js. Para establecer esta comunicación creamos por un lado una aplicación django y por otro una aplicación react en el servidor de node.js. La aplicación react fue configurada con Webpack, una herramienta de agrupación de módulos. Una vez hecho esto, gracias a django webpack_loader y webpack-bundle-tracker conseguimos la comunicación deseada. De esta

forma cuando el navegador pide a Django el cliente, este desvía la petición al puerto 3000, donde se encuentra node.js y este se lo devuelve renderizado. Esto será explicado en más detalle en el apartado de implementación (capítulo 6).

Estas aplicaciones comparten una funcionalidad común: visualización de la evolución de los pacientes hospitalizados (ver figura 6.1). Estos datos se ofrecen en modo de tabla y en representación gráfica tal y como se puede observar en la imagen. Esto último se hace con la finalidad de ofrecerle al personal sanitario la posibilidad de observar rápidamente si la evolución de su paciente es positiva o negativa.

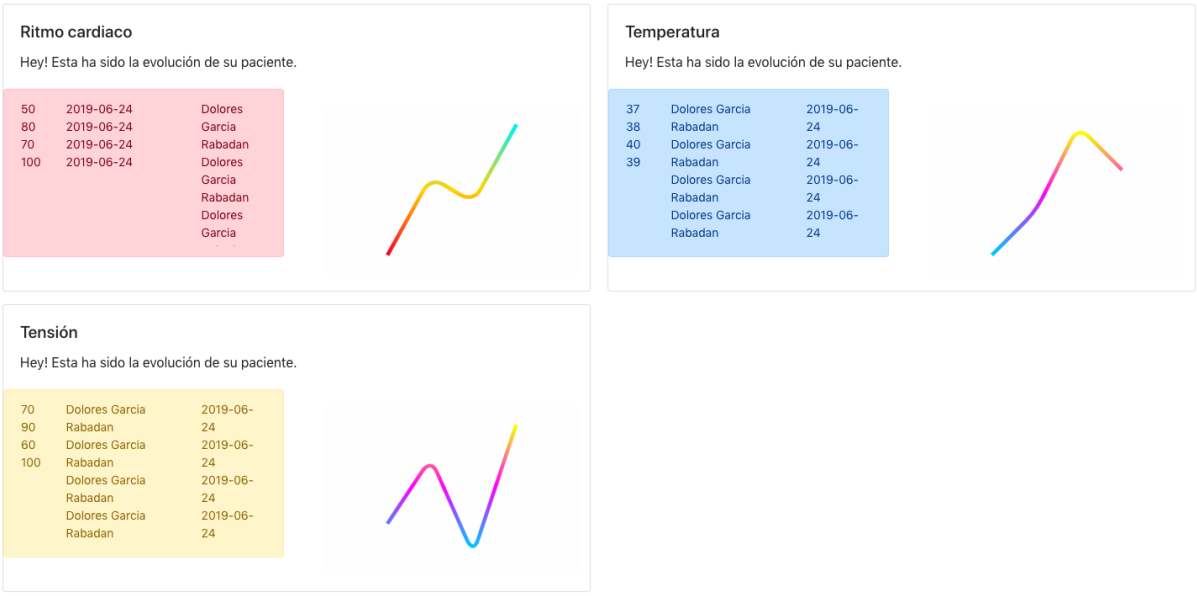

Figure 6.1: Evolución del paciente

Por otro lado, la aplicación de enfermería (mostrada en la figura 6.5) tiene tres funcionalidades más añadidas:

1. **Creación de alarmas:** Aquí el usuario podrá crear alertas relativas a pacientes que considere importantes a conocer por el personal sanitario que le siga en el siguiente turno. Para ello la aplicación le facilitará un formulario (figura 6.2) donde

el usuario deberá seleccionar el paciente, describir la alerta y indicar si es urgente o no. Posteriormente esta información será enviada a nuestra base de datos mediante una petición post.

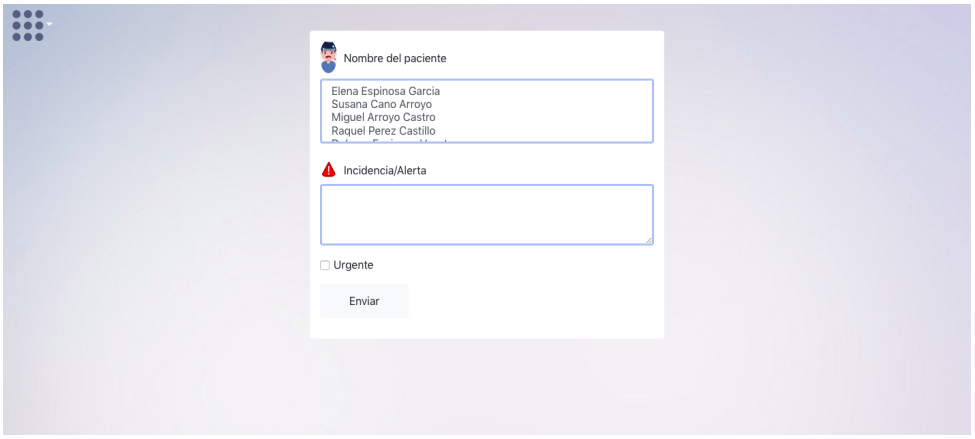

Figure 6.2: Creación de alertas

2. **Visualizar agenda de pacientes:** Esta opción (figura 6.2) muestra al usuario dos paneles, uno de ellos con los pacientes ya visitados y otro con los no visitados. Dentro de ellos, los pacientes aparecen como *checkboxs*, permitiendo así que el usuario los mueva de un panel a otro.

   Además al lado de cada paciente aparecerá una campana de color amarillo si este presenta alertas. Si es así, el usuario podrá hacer *click* en la campana y el sistema lo redirigirá al historial del paciente, donde aparecen todas las alertas del enfermo.

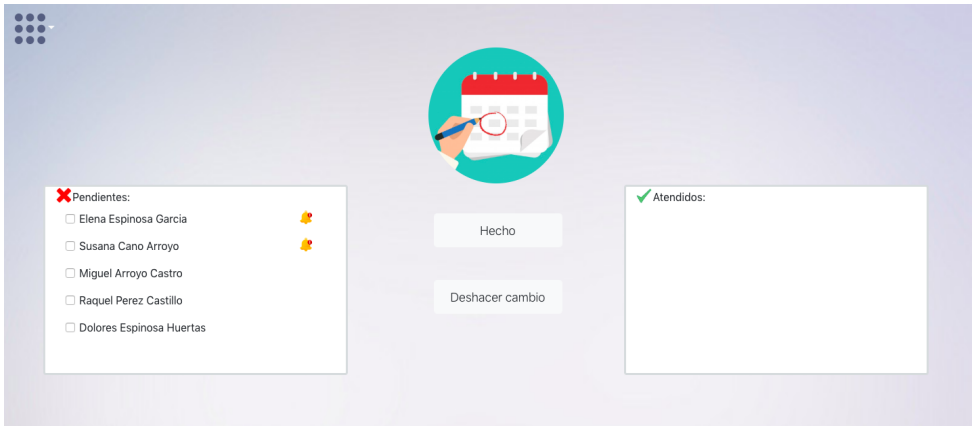

Figure 6.3: Agenda

3. **Conexión con el dispositivo bluetooth** Al seleccionar esta opción se abrirá una ventana con un scanner para la identificación del paciente. Tras la identificación de este el sistema establecerá una conexión BLE con el dispositivo bluetooth y mostrará al enfermero o auxiliar el valor recogido y lo enviará al historial médico del paciente. El sistema también permitirá al usuario repetir la medición.

   La conexión con el servidor GATT se realiza mediante una conexión bluetooth low energy. BLE Esta conexión se lleva a cabo mediante la Web Bluetooth API[4], una api Javascript diseñada para la conexión de sitios web con dispositivos bluetooth sin la necesidad de usar aplicaciones nativas. Una vez establecida la conexión se realiza el intercambio de información de forma bidireccional entre el dispositivo periférico y el dispositivo central, siendo el dispositivo medidor el periférico y el dispositivo central el móvil, ordenador o tablet donde se esté ejecutando la aplicación. Este intercambio de información se realiza mediante la definición de servicios y características. Los servicios y características nos permiten definir el tipo de información que se intercambian los dispositivos. Los servicios se utilizan para dividir las características en entidades lógicas, por ejemplo: el servicio de temper-

atura (health_thermometer) agrupa las características: temperature_measurement, intermediate_temperature, temperature y temperature_type. Estos son identificados mediante un UUID, que en caso de ser estándar está definido por 16 bits y en caso de ser personalizados es de 128 bits. En este caso nosotros hemos utilizado los servicios y características estándar, es decir los definidos por bluetooth.
En este caso los servicios seleccionados fueron:

- **health_thermometer**: medicición de la temperatura
- **heart_rate**: medición de la presión arterial.
- **blood_pressure**: medición de la frecuencia cardiaca.

y las características:

- **temperature_measurement**: valor de la temperatura.
- **heart_rate_measurement**: valor de la presión arterial media.
- **blood_pressure_measurement**: valor de la frecuencia cardiaca.

Una vez definidos los servicios y características se procede a la selección de la función que deseemos realizar. Estas son:

- Indicate
- Read
- Write

De estas tres nosotros solo usamos la función: read, pues solo necesitábamos obtener los valores de las constantes para enviarlas al historial médico mediante una petición post y posteriormente recuperarlas para su visualización.

Esta comunicación periférico-central en Bluetooth Low Energy (Bluetooth 4.0) se realiza bajo dos protocolos, protocolo GATT (Generic Attribute Profile) Y protocolo ATT (Attribute Protocol). El protocolo ATT se encuentra en un nivel inferior al protocolo GATT6.4 y este es utilizado para almacenar servicios y características en forma de tabla. El protocolo GATT por otro lado va a definir como se realiza el intercambio de información utilizando servicios y características. Cabe destacar que el periférico es el que contiene los datos de búsqueda de ATT y las definiciones de servicios y características6.4, y el central, el que envía peticiones a este. Es por ello que denominamos al dispositivo periférico servidor GATT o dispositivo esclavo y al dispositivo central cliente GATT.

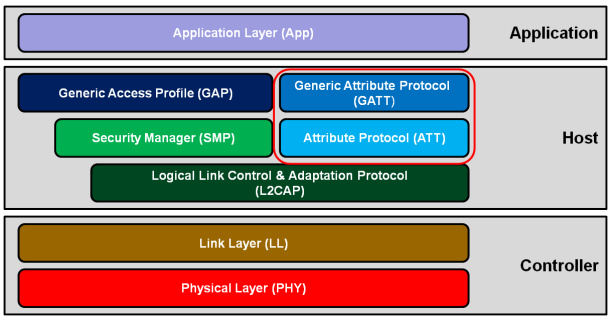

Figure 6.4: Protocolos GATT y ATT

Respecto al sistema de almacenamiento, hemos utilizado una base de datos sql y para su gestión utilizamos el gestor de base de datos sqlite3. Aunque este intercambio de información no se realizó de forma directa, pues utilizamos los modelos de Django como intermediarios. Los modelos de Django implementan un ORM (object-relational mapping) que nos permite interaccionar con la base de datos sin necesidad de utilizar el lenguaje SQL. Esto nos ha permitido realizar un intercambio de información entre sqlite3 y la aplicación de forma sencilla y rápida.

Por último cabe destacar que utilizamos un simulador virtual para la simulación de los dispositivos medidores. Estos los creamos gracias a la aplicación: LightBlue[1] que permite la creación de periféricos Bluetooth Low Energy virtuales con los servicios y características que les indiquemos, que en este caso fueron las mencionas anteriormente.

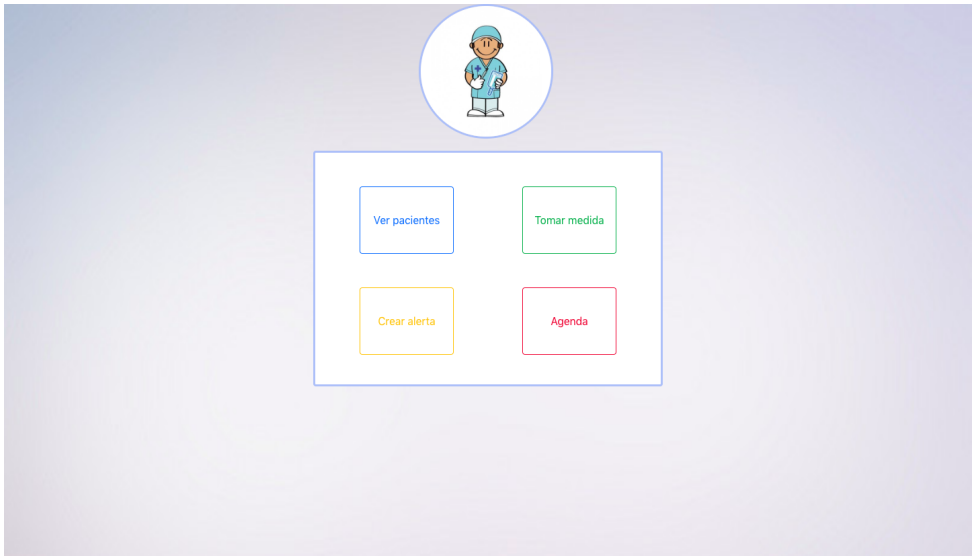

Figure 6.5: Página principal aplicación enfermería

[1]LightBlue App https://apps.apple.com/es/app/lightblue-explorer/id557428110
https://play.google.com/store/apps/details?id=com.punchthrough.lightblueexplorerhl=en$_US$

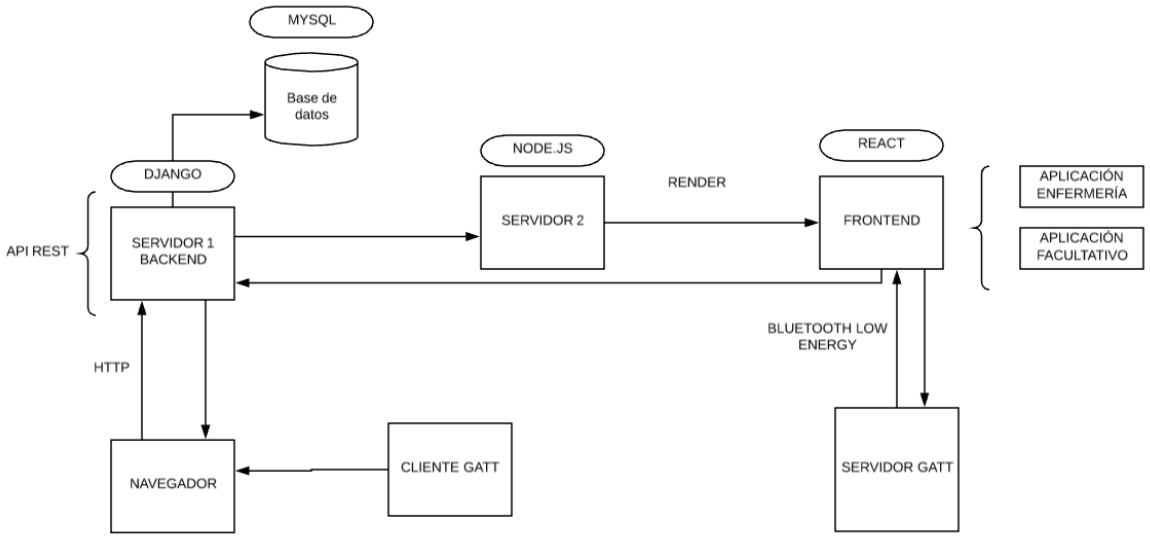

Figure 6.6: Big Picture

Chapter 7

Implementación

En el capítulo anterior describimos la arquitectura de nuestro proyecto, y sentamos las bases de la implementación de nuestra aplicación.

En este capítulo describiremos con más en detalle la implementación de algunas secciones y componentes clave de la aplicación relativas al front-end.

7.1 Conexión con el dispositivo periférico

Para llevar a cabo la conexión con los dispositivos periféricos hemos implementado cuatro componentes.

A continuación describiremos cada componente por separado.

7.1.1 Componente Bluetooth

Este componente es el encargado de realizar la conexión con los distintos dispositivos bluetooth. Este se activa cuando el usuario pulsa la opción tomar medición y comienza a buscar dispositivos BLE que posean los servicios que le hemos indicado: heart_rate,

blood_presure y health_thermometer.

Como se muestra a continuación la función que permite el establecimiento de la conexión BLE es navigator.bluetooth.requestDevice() que recibe como argumento los distintos servicios a los que deseamos conectarnos.

```
navigator.bluetooth.requestDevice(
{acceptAllDevices: true, optionalServices: ['heart_rate',
'health_thermometer','blood_pressure', ]})
```

El diseño de este componente le permite identificar el tipo de dispositivo al que se ha conectado mediante la obtención del nombre del dispositivo, el cual es elegido por el fabricante, que en este caso, al ser dispositivos periféricos simulados, somos nosotros. En este caso no deseamos obtener servicios que pudiesen ser comunes a los tres periféricos, como podría ocurrir con el nivel de batería, por lo que al identificar el nombre del dispositivo BLE podemos pedir el servicio mediante la función:

```
server.getPrimaryService([servicio])
```

La finalidad de determinar primero el tipo de dispositivo al que nos estamos conectando es evitar que el usuario tenga que tenga que indicar que tipo de medición quiere realizar y permitir así que sea el sistema el que lo determine de forma automática, limitando la interacción del usuario con la aplicación a:

1. Indicar si desea realizar medición
2. Indicar si desea Enviar medición
3. Indicar si desea Repetir medición

Una vez conectado al dispositivo bajo el servicio correspondiente, Bluetooth llamará a los componentes: HeartRate, BloodPresure o Temperature en función de cual sea este servicio.

7.1.2 Componente HeartRate

El componente HeartRate como su nombre indica es el encargado de realizar la medición del ritmo cardiaco del paciente. Este recibe el objeto servicio del componente bluetooth y comienza la lectura bajo la característica:

- **heart_rate_meassurenmet:** representa el valor medio de la frecuencia cardiaca del paciente.

La obtención de las características se realiza de la siguiente forma:

```
service.getCharacteristic({heart_rate_meassurenmet)
service.getCharacteristic(heart_rate_control_point)
```

Una vez realizada la medición el sistema leerá el resultado de la lectura mediante la siguiente función:

```
characteristic.readValue()
```

y mostrará al usuario una pequeña ventana con el valor recogido (figura 7.4). El usuario también podrá repetir la medición pulsando el botón: repetir medición. Cuando el usuario pulse esta opción el sistema volverá a leer el valor de la frecuencia cardiaca sin establecer una nueva conexión con el dispositivo bluetooth y mostrará al usuario el nuevo valor.

Si el usuario pulsa la opción:enviar y los datos se envían correctamente, el sistema indicará al usuario que la medición se ha anotado de forma correcta (figura 7.2), en caso de no ser así el sistema llevará al usuario a una ventana de error (figura 7.3). Sea cual sea el caso, posteriormente, el sistema redirigirá al usuario de nuevo al menú principal.

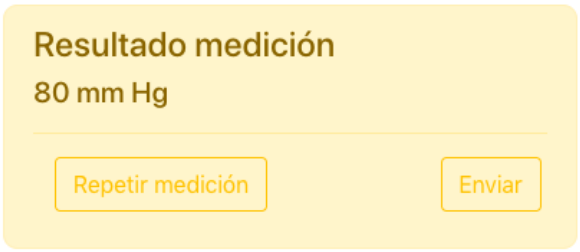

Figure 7.1: Resultado medición presión arterial

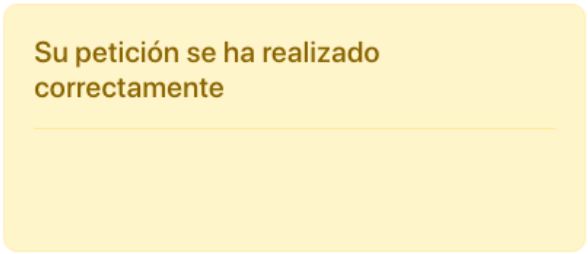

Figure 7.2: Mensaje de confirmación

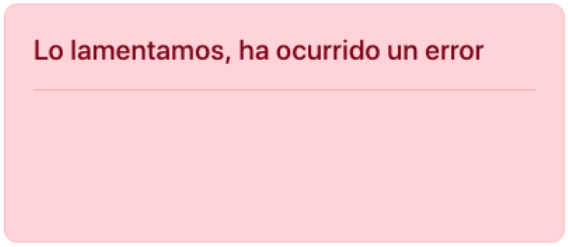

Figure 7.3: Mensaje de error

7.1.3 Componente BloodPresure

Este componente tiene como finalidad determinar la tensión del paciente. Este al igual que el componente HeartRate se conecta al servicio dado por el componente Bluetooth y realiza la comunicación con el dispositivo BLE. En este caso, para la característica:

- **blood_pressure_measurement** : indica el valor de la presión arterial media.

La forma de obtención de las características y del valor medido son mediante las dos funciones descritas en el componente: HeartRate. También, como en el caso de la frecuencia cardiaca, el sistema mostrará al usuario una ventana con el resultado obtenido y permitirá a este volver a repetir la medición. El sistema como en el caso anterior hará saber al usuario si la anotación se ha realizado de forma correcta (pantallas 7.2 y 7.3).

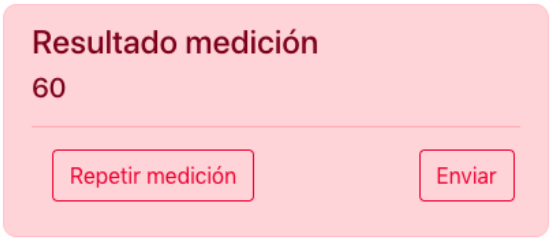

Figure 7.4: Resultado medición de la presión arterial

7.1.4 Componente Temperature

Por último, el componente **temperature**. Este componente determina la temperatura del paciente. En este nos conectaremos al periférico para la característica:

- **temperature_meassurenmen**: determina el valor de la temperatura

Como en el caso anterior, no volveremos a mostrar las funciones que permiten la obtención de la característica y la lectura del valor, pues ya han quedado descritas. Como en los dos casos anteriores, el sistema mostrará al usuario el resultado de la medición en la ventana emergente y permitirá a este volver a repetir la medición. El sistema también mostrará al usuario si la anotación de la temperatura se ha realizado de forma correcta (pantallas 7.2 y 7.3).

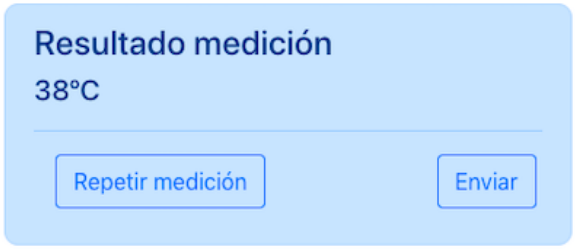

Figure 7.5: Resultado medición temperatura

Aunque a simple vista puede parecer que estos tres componentes comparten muchas funcionalidades comunes y que podían haber sido simplificadas en un solo componente, la separación de estos en distintos componentes, no sólo aporta una mejor legibilidad del código, también permite que el sistema sea más flexible a la incorporación de nuevos servicios así como de nuevas características a los servicios ya existentes. Esto último, debido al uso de promesas en la implementación haría que el código difiriese considerablemente de un componente a otro.

7.2 Gestión de pacientes

Para esta sección vamos a dar breves pinceladas sobre parte de la implementación de los distintos componentes que nos han permitido llevar a cabo la gestión de pacientes.

Existen más componentes además de estos seis. Estos nos han permitido la reutilización del código en estos cinco componentes principales y los nombraremos en cada sección si corresponde.

7.2.1 Componente Main_Nurse

Este componente implementa la pantalla principal6.5 de los usuarios enfermeros o auxiliares de enfermería y realiza el redireccionamiento a los distintos componentes que implementan las funcionalidades presentes en la pantalla principal y con las que el usuario interactúa. De esta forma, cada vez que el usuario hace **click** en alguno de los cuatro botones que componen el menú principal, el componente Inicio realiza el redireccionamiento al componente correspondiente mediante la siguiente función:

```
renderRedirect = () => {
    if (this.state.redirect_agenda) {
      return <Redirect to={{pathname: '/planner/',state:
      { id: this.state.id }}}/>
    } else if(this.state.redirect_alerta){
      return <Redirect to={{pathname: '/form_alert/',state:
      { id: this.state.id }}}/>
    } else if(this.state.redirect_medida){
      return <Redirect to={{pathname:
      '/login_pac/',state:
      { id: this.state.id }}}/>
```

```
    } else if(this.state.Redirect_pacientes){
      return <Redirect to={{pathname: '/list_patients/',state:
      { id: this.state.id , user: 'nurse'}}}/>
    } else if(this.state.cerrar_sesion){
      return <Redirect to={{pathname: '/login/'}}/>
    }
  }
```

7.2.2 Componente Planner

Este componente implementa una ventana donde el usuario enfermero o auxiliar de enfermería podrá listar sus pacientes y ver si estos tienen alarma6.3. Estos pacientes y alarmas se obtienen mediante una petición post que realizamos a nuestra API Rest, implementada en Django, de la siguiente forma:

```
axios.get('/blog/Patient/', {
    params: {
        id: this.props.location.state.id
    }
})
.then(response =>{
    this.setState({ patients: response.data['patients']});
    this.setState({ patient: response.data['patients']});
    this.setState({ alertas: response.data['alerts']});
    this.setState({ id_patient: response.data['ids']});
})
```

De esta forma conseguimos el nombre de nuestros pacientes, sus identificadores y sus alertas.

Como se explicó en el diseño, esta alarma queda representada mediante una pequeña campana amarilla al lado del nombre del paciente y si el usuario hace **click** en ella el sistema redireccionará al usuario al historial médico de ese paciente, donde podrá ver estas alarmas.

Por último cabe mencionar que esta aplicación llama al componente: Go [no mostrado] que implementa un Dropdowns que realiza el redireccionamiento a otras páginas en función de la opción que seleccione el usuario. Este componente es llamado por todos los componentes de la aplicación, aunque no se mencione, pero para cada uno muestra funcionalidades distintas.

7.2.3 Componente FormAlert

Este componente es el que permite que los usuarios enfermeros o auxiliares de enfermería6.2 puedan registrar alarmas relativas a pacientes. Este componente por tanto debe realizar una petición get a nuestra API Rest para pedir los pacientes asociados a ese enfermero y una petición post para enviar los datos del formulario a la base de datos.

```
axios.get('/Patient/', {
    params: {
        id: this.props.location.state.id
    }
})
.then(response =>{
    this.setState({ patients: response.data['patients']});
})
```

```
axios.post('/Save_Alert/', incident)
```

El sistema hará ver al usuario si la alarma se ha enviado correctamente o no 7.2 7.3, como ocurre en el caso de la anotación de las constantes vitales.

7.2.4 Componente List_patients

Este componente es el que da paso al historial médico del paciente donde se muestra la información relativa a constantes y sus alarmas. Este componente muestra una lista 7.6 con pacientes, para lo cual, como en los casos anteriores, tiene que realizar una petición post a nuestra API Rest. De esta forma, si el usuario hace **click** en uno de ellos el sistema lo redireccionará a los componentes: NurseHistory, diseñado para enfermeros o MedicineHistory, diseñado para facultativos donde se mostrará el historial médico del paciente. La razón de separar estos dos componentes se explicará en la siguiente sección. Este componente es por tanto común a médicos y enfermería. Los facultativos accederán a el desde la página de inicio de sesión y la enfermería desde el menú principal cuando seleccionan la opción ver pacientes. Así, este componente distingue el usuario desde el que se ha accedido a el y en función de ello cuando el usuario seleccione el paciente a visitar el sistema lo redireccionará a un componente u a otro.

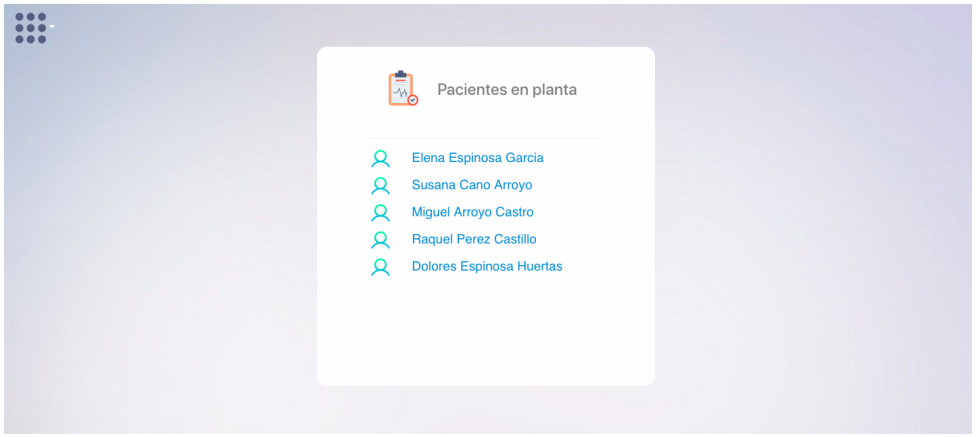

Figure 7.6: Resultado medición temperatura

7.2.5 Componentes NurseHistory y MedicineHistory

Estos componentes como ya hemos mencionado anteriormente son los que implementan el historial médico del paciente 6.1. Estos no tienen las mismas funcionalidades para la enfermería y los facultativos, pues a los médicos solo debe mostrarles información relativa a constantes y a la enfermería debe mostrarle también las alertas relativas a ese paciente 7.7. Por ello se implementó otro componente: VitalSign. Este componente es el que implementa la parte del historial que representa la información de constantes vitales de pacientes y es llamado por los componentes: NurseHistory y MedicineHistory gracias a que React nos permite realizar esta reutilización de los componentes.

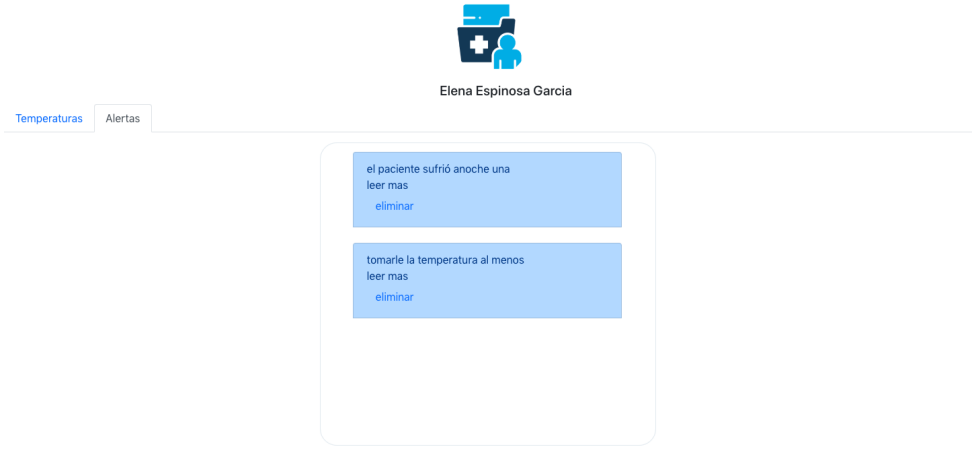

Figure 7.7: Alertas paciente seleccionado

7.3 Autenticación del personal sanitario y del paciente

La autenticación del personal sanitario y del paciente la realizan los componentes: Scanner y Login respectivamente.

El Login formulario implementa una ventana donde se muestra un pequeño formulario 7.8 con los campos: usuario y contraseña. Este componente enviará un post a django con los valores de los campos del formulario para validar al usuario. Si estos son correctos django responderá con un ok y el componente Login realizará un rediccionamiento al componente Main_Nurse, en caso de que el usuario sea enfermero o al componente List_patients en caso de que el usuario sea el facultativo.

El componente **Scanner** por otro lado7.9, facilita la lectura del código de barras de la pulsera del paciente cuando el enfermero pulsa la opción: ***tomar medida***, presente en el menú principal. La lectura del código de barras se ha realizado gracias a la función: QrReader de la librería *react-qr-scanner*. Este componente sigue el mismo procedimiento

que el componente Login_PS, realiza un post a la api rest con los datos recogidos para su validación y si estos son correctos realiza un redireccionamiento en este caso al componente **bluetooth**. Este redireccionamiento se realizará cuando el usuario pulse el botón iniciar medición, que se hará visible cuando el sistema valide al paciente 7.10. Además de habilitar el botón de inicio de medición, el sistema emitirá un **beep** para informar al usuario de que la autenticación del paciente se ha realizado de forma correcta.

En ambos casos, si el sistema no reconoce como usuario del sistema al personal sanitario o al paciente emitirá un mensaje de error.

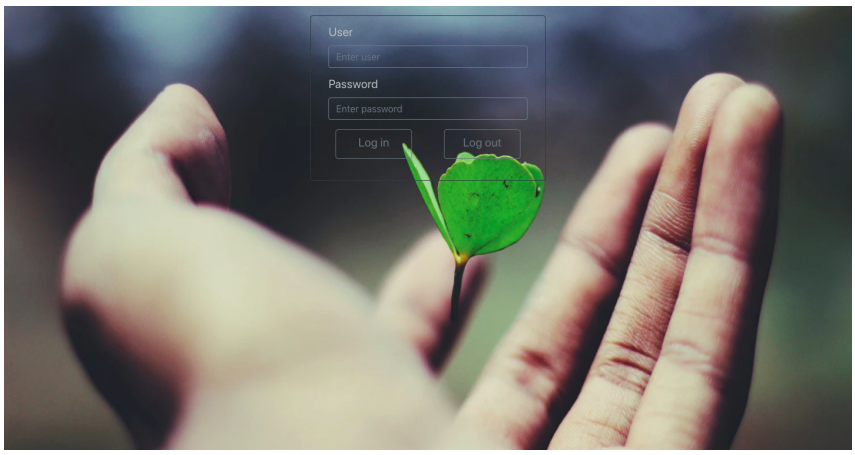

Figure 7.8: Ventana de inicio de sesión

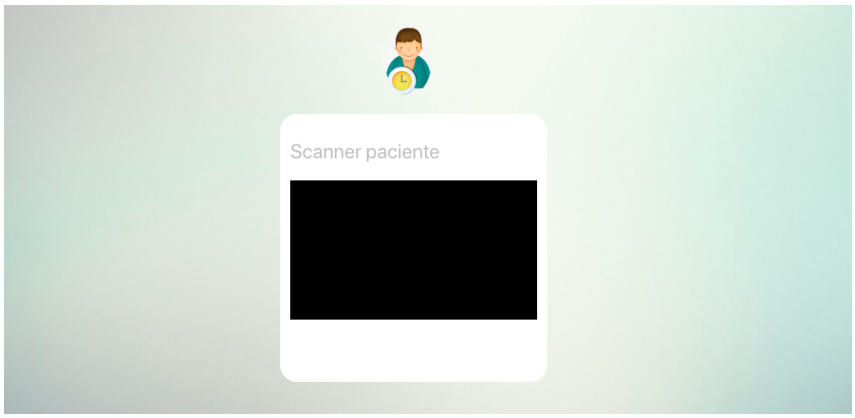

Figure 7.9: Autenticación paciente

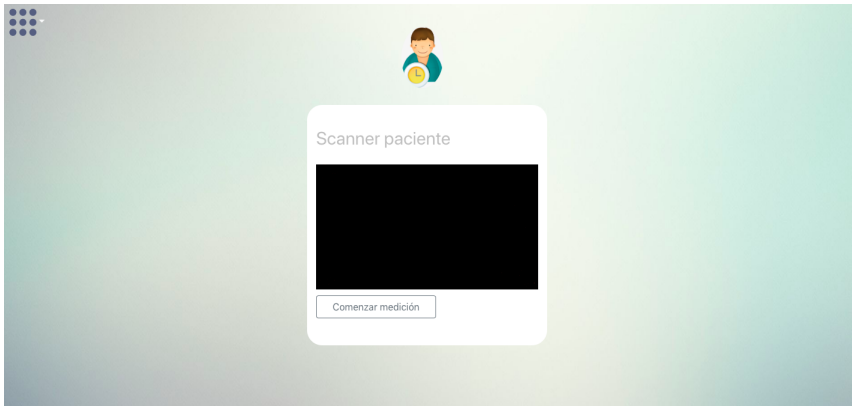

Figure 7.10: Comenzar medición

7.4 Cambio de contraseña

El componente que implementa el cambio de contraseña es: NewPassword. Este componente muestra al usuario un pequeño formulario 7.11 que tras ser rellenado será enviado a la base de datos mediante una petición POST. Cabe destacar que el sistema sólo permitirá al usuario cambiar su contraseña; el usuario no podrá cambiar el identificador de usuario. Una vez enviado el formulario, el sistema hará saber al usuario si el cambio se ha realizado de forma correcta. Si es así, el sistema llevará al usuario a una nueva ventana donde se le indicará que sus cambios se han realizado correctamente (Figura 7.2). De no ser así el sistema mostrará los bordes del formulario en color rojo (pantalla 7.12).

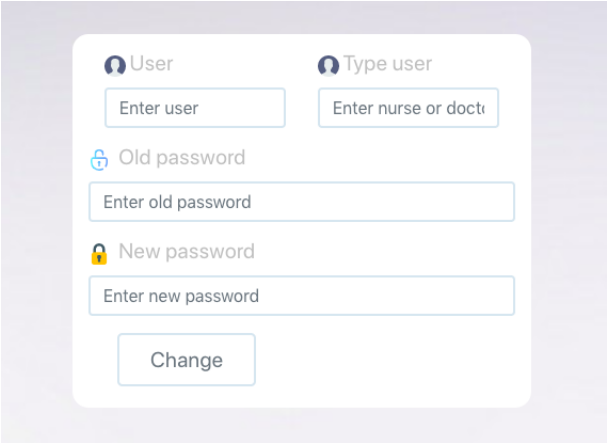

Figure 7.11: Formulario cambio de contraseña

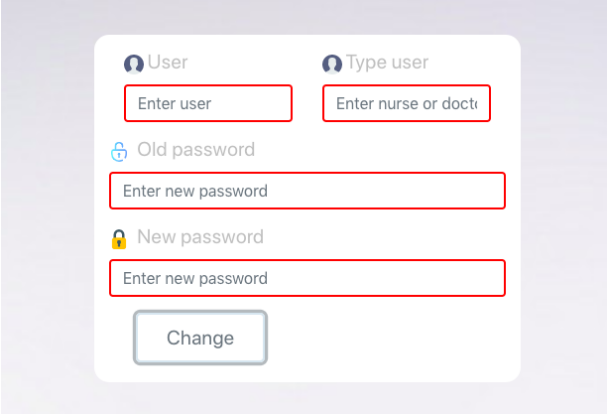

Figure 7.12: Error cambio de contraseña

7.5 Renderización de React en el Servidor

Como ya mencionamos en el capítulo anterior, la renderización de React en el servidor se realizó en node.js y gracias a django webpack_loader y webpack-bundle-tracker conseguimos la comunicación entre nuestra api rest y node.js. Estas tecnologías nos han permitido establecer las configuraciones necesarias para comunicar el servidor de Django y el servidor de node.js de forma que cuando llegue una petición a django, este la desvíe

al puerto 3000, donde se renderizará el cliente que será enviado de nuevo al puerto 8000 ya cargado.

Para establecer esta comunicación creamos por un lado una aplicación django y por otro una aplicación React en el servidor de node.js. Así cuando llega una petición al servidor de django este trata de renderizar un html que posee la siguiente estructura:

```
{% load render_bundle desde webpack_loader%}
 <! DOCTYPE html>
 < html >
   < cabeza >
     < meta charset = "UTF-8" />
     < meta name = "viewport" content = "width = device-width" />
     < título > Ponynote </ título >
   </ cabeza >
   < cuerpo >
     < div id = "root" >
     </ div >
       {% render_bundle 'main'%}
   </ cuerpo >
 </ html >
```

la etiqueta "root" representa el punto de inicio de nuestra aplicación react. Por otro lado la etiqueta render_bundle con main como argumento representa la etiqueta de script para el paquete denominado main que es producida por la configuración que hemos establecido con webpack. Así cuando el cliente realiza una petición al servidor de django, este renderiza el html anterior, el cual lleva al script *jsx raiz* de React que tiene el siguiente aspecto:

```
ReactDom.render(
    <Router>
    <App />
</Router>,
document.getElementById('root')
);
serviceWorker.unregister();
```

Así mismo este invoca al componente App, que gracias a la librería *react-router-dom* realiza el enrutamiento permitiéndonos así cargar en el servidor de node todos los componentes anteriormente descritos.

Con respecto al cliente, como hemos mencionado al principio, hemos mostrado solo algunos de los componentes más esenciales, pues en total son 24. La descripción del código es también bastante breve, por ello en el anexo mostraremos la implementación completa de alguno de los componentes. Por otro lado, con respecto al servidor de Django, el modelo y la vista tampoco han sido descritos en este capítulo pero si serán adjuntados en el anexo.

Chapter 8

Evaluación del impacto

Este proyecto supone un cambio importante en la forma de trabajo en el área de hospitalización de un hospital. Esta aplicación pretende eliminar las monótonas y tediosas gestiones administrativas manuales en las que se ven involucrados enfermeros, auxiliares y médicos en su día a día y así simplificar su trabajo a labores meramente clínicas. Este trabajo supone la automatización de un trabajo que desde siempre se ha realizado a mano. Esto no solo supone un cambio en la forma de anotación de constantes, si no que también genera un cambio en la forma de comunicación del personal sanitario, que ahora lo hará de forma telemática mediante las web apps gracias a la posibilidad de visualizar la evolución del paciente mediante el inicio de sesión en la aplicación. Darle al personal sanitario la posibilidad visualizar la evolución de sus pacientes en turnos previos o de crear y visualizar alarmas relativas a pacientes permite establecer una comunicación multidireccional entre el persona sanitario, lo que se traduce en un personal sanitario mejor informado y por tanto con la capacidad de tomar mejores decisiones. La privacidad del paciente también se ve afectada positivamente pues la eliminación del papel, y su paso a un soporte digital supone un control del acceso a sus datos sensibles, garantizando así la protección de su intimidad, que ahora hasta ahora podía verse comprometida.

Así podríamos decir que esta aplicación genera un impacto no solo tecnológico, también social, que en un futuro podría llegar a todas las áreas del hospital.

Chapter 9

Líneas futuras

Aunque nuestra aplicación pretende una automatización del proceso de anotación de constantes, esta no consigue una automatización total, pues hay muchas funcionalidades de la aplicación que siguen requiriendo la intervención manual del personal sanitario. Comenzaremos con la autenticación del paciente. El paciente es identificado mediante la lectura de un código de barras que posee en su pulsera. Esta autenticación esperamos que en un futuro muy próximo se realice de forma automática como ocurre con la tecnología B Anethesis, donde el usuario es identificado en quirófano de forma automática mediante la conexión que se establece entre su pulsera y la aplicación. Otro aspecto mejorable es la simplificación de todos los dispositivos BLE que hemos utilizado (en este caso simulados) en un único periférico que actúe a modo de sensor y nos permita obtener todas las constantes vitales del paciente sin la necesidad de intervención o al menos reduciéndola, del enfermero o auxiliar. También la base de datos implementada es una simplificación de la realidad que nos ha servido para poner en marcha la aplicación. Pues por ejemplo para el componente Planner se espera la implementación de un superusuario, que permita al jefe de enfermería asociar los pacientes a cada turno. Por último, referente a la creación de alarmas, la opción boleana: urgente, se ha implementado con la intención de que en una optimización de esta aplicación el sistema muestre ventanas emergentes al usuario

enfermero o auxiliar de enfermería tras entrar estos en el menú principal si alguno de sus pacientes tienen alarmas catalogadas como urgentes. Esperamos que estas limitaciones sirvan como aliciente para seguir avanzando en este área y conseguir una automatización total del proceso.

Chapter 10

Conclusiones finales

Hemos desarrollado dos aplicaciones web, una destinada a la enfermería y otra a facultativos, para el área de hospitalización.

La aplicación destinada a enfermería es capaz de establecer una conexión bluetooth low energy con dispositivos de medición de constantes vitales que posean los servicios: health_thermometer (temperatura corporal), blood_pressure (presión sanguínea) y heart_rate (frecuencia cardiaca), y leer los resultados de las mediciones realizadas por estos periféricos bajo las características: temperature_measurement, heart_rate_measurement y blood_pressure_measurement. Esta aplicación también es capaz de llevar a cabo una gestión de pacientes mediante la creación de alarmas, la disposición de una agenda de pacientes y la visualización de la evolución de sus pacientes.

Esta última funcionalidad es compartida por la aplicación de los facultativos, que ahora podrán llevar un seguimiento de la evolución de sus pacientes de forma telemática desde cualquier punto a través de un navegador Web.

Bibliography

[1] Mensoft. Hydra: Tecnologías efectivas para la monotorización. http://mensoft.es/company/hydra/.

[2] Vincenzo Auletta, Carlo Blundo, Emiliano De Cristofaro, and Guerriero Raimato. Web services invocation over bluetooth. *Wireless Sensor Network*, 2:447–461, 01 2010.

[3] Jordán Pascual Espada, Vicente García Díaz, Rubén González Crespo, Oscar Sanjuán Martínez, B Cristina Pelayo G-Bustelo, and Juan Manuel Cueva Lovelle. Using extended web technologies to develop bluetooth multi-platform mobile applications for interact with smart things. *Information fusion*, 21:30–41, 2015.

[4] Francois Beaufort. Interact with bluetooth devices. https://developers.google.com/web/updates/2015/07/interact-with-ble-devices-on-the-web.

[5] Carles Gomez, Joaquim Oller, and Josep Paradells. Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology. *Sensors*, 12(9):11734–11753, 2012.

[6] Matti Siekkinen, Markus Hiienkari, Jukka Nurminen, and Johanna Nieminen. How low energy is bluetooth low energy? comparative measurements with zigbee/802.15.4. *IEEE Wireless Communications and Networking Conference Workshops (WCNCW), Paris, France*, 04 2012.

[7] Kappta. Cómo mejorar la atención al paciente mediante el uso de beacons. https://blog.kappta.com/wp-content/uploads/2016/06/cropped-favicon-270x270.png.

[8] Kappta. Ble. https://blog.kappta.com/ble-o-bluetooth-low-energy/.

[9] Using Beacons. Múltiples ventajas de la tecnología beacon para la industria de la salud. https://blog.kappta.com/wp-content/uploads/2016/06/cropped-favicon-270x270.png.

[10] Uniwebsidad. Python para principiantes. https://uniwebsidad.com/libros/python.

[11] Uniwebsidad. El libro de django 1.0. https://uniwebsidad.com/libros/django-1-0.

[12] Mozilla developers. Framework web django (python. https://uniwebsidad.com/libros/django-1-0.

[13] Django web official. Django documentation. https://docs.djangoproject.com/en/2.1/.

[14] scotch.io. Build a to-do application using django and react. https://scotch.io/tutorials/build-a-to-do-application-using-django-and-react.

[15] Will Vincent. Django rest framework with react tutorial. https://wsvincent.com/django-rest-framework-react-tutorial/.

[16] React official page. A javascript library for building user interfaces. https://reactjs.org/.

Appendix A

Implementación

A.1 Componente Bluetooth

```
export default class bluetooth extends React.Component {
  state = {
    value : '',
    service: '' ,
    temperature: false,
    heart_rate:false,
    blood_pressure: false,
    idenf: ' ',
    device: '',
    error_event: false
  }
  componentWillMount(){
    navigator.bluetooth.requestDevice(
      {acceptAllDevices:true,
      optionalServices: ['heart_rate', 'health_thermometer',
      'blood_pressure', ]})
      .then(device => {
        console.log('Connecting to GATT Server...');
        console.log(device.gatt.connect())
        this.setState({device: device})
        return device.gatt.connect();
      })
      .then(server => {
        if(server.device.name=="Health Thermometer"){
         this.setState({ service: server.getPrimaryService
```

```
        ('health_thermometer')})
        this.setState({ temperature:true})
      }
      else if(server.device.name=="Heart Rate"){
        this.setState({ service: server.getPrimaryService
        ('heart_rate')})
        this.setState({ heart_rate:true})
      }

      else if(server.device.name=="Blood Pressure"){
        this.setState({ service: server.getPrimaryService
        ('blood_pressure')})
        this.setState({ blood_pressure:true})
      }
    })
    .catch(error => {
      this.setState({error_event: true})
    })
}
Error = () =>{
  if(this.state.error_event){
    return <Redirect to={{pathname: '/error/',state: {id:
    this.state.this.props.location.state.id_e}}}/>
  }
}
```

```
render() {
    return (
      <div className="TemperaturaB" >
        {this.state.temperature &&
          <Redirect to={{pathname: '/obtain_temperature/', result:
          { service:this.state.service, tempdevice:this.state.device,
          characteristic:"temperature_measurement",
          id_nurse: this.props.location.state.id_e, id_patient:
          this.props.location.state.id_patient}}}/>}

        {this.state.blood_pressure && <Redirect to={{pathname:
        '/obtain_blood_pressure/', result:{ service:
        this.state.service, pressuredevice:this.state.device,
        characteristic:"blood_pressure_measurement",
        id_nurse: this.props.location.state.id_e
        ,id_patient: this.props.location.state.id_patient}}}/>}

        {this.state.heart_rate && <Redirect to={{pathname:
        '/obtain_heart_rate/', result:{ service:this.state.service,
        heartdevice:this.state.device, characteristic:
        "heart_rate_measurement", id_nurse: this.props.location.state.id_e ,
        id_patient: this.props.location.state.id_patient}}}/>}
      {this.Error()}
      </div>
    )
  }
```

```
}
```

A.2 Componente Temperature

```
axios.defaults.xsrfCookieName = 'csrftoken'
axios.defaults.xsrfHeaderName = 'X-CSRFToken'
export default class Temperature extends React.Component {
  state = {
    value: '',
    error_event: false,
  }
  componentWillMount() {
    this.props.location.result.service.then(service => {
      return service.getCharacteristic(this.props.location.result
      .characteristic)
   })
   .then(characteristic => {
      return characteristic.readValue();
   })
   .then(value => {
      this.setState({value: value.getUint8(0)})
   })
    .catch(error => {
      this.setState({error_event: true});
    });
  }
```

```
Repeat = () => {
  this.props.location.result.tempdevice.gatt.connect()
  .then(server => {
    return server.getPrimaryService('health_thermometer')})
  .then(service => {
    return service.getCharacteristic(this.props.location.result.
    characteristic)
  })
  .then(characteristic => {
    return characteristic.readValue();
  })
  .then(value => {
    this.setState({value: value.getUint8(0)})
  })
  .catch(error => {
    this.setState({error_event: true})
  });
}
Send = () => {
  const temperature = {
    value: this.state.value,
    patient: this.props.location.result.id_patient,
    idnurse: this.props.location.result.id_nurse
  };
  axios.post('/save_temperature/', temperature)
  .then(res => {
    if(res.data=="ok"){
```

```
        this.setState({activate_event:true})
      }
      else{
        this.setState({error_event: true})
      }
    })
  }
  Redirect_sent = () =>{
    if(this.state.activate_event){
      return <Redirect to={{pathname: '/sent/',state: {id:
      this.props.location.result.id_nurse}}}/>
    }
  }
  Error = () =>{
    if(this.state.error_event){
      return <Redirect to={{pathname: '/error/',state:
      {id: this.state.idn}}}/>
    }
  }
  render() {
    return (
      <div className="Temperature">
        <Col xs={6} sm={6} md={6} lg={6}>
          <Alert variant="primary"
            style={{
              width: 400,
              height: 170,
```

```
              borderRadius: 10,
              marginTop: 25
            }}>
            <Alert.Heading>Resultado medición</Alert.Heading>
              <h5>{this.state.value}°C</h5>
            <hr />
            <div className="d-flex justify-content-end">
              <Col xs={9} sm={9} md={9} lg={9}>
                <Button variant="outline-primary" onClick={this.Repeat}>
                  Repetir medición
                </Button>
              </Col>
              <Col xs={3} sm={3} md={3} lg={3}>
                <Button variant="outline-primary" onClick={this.Send}>
                  Enviar
                </Button>
              </Col>
            </div>
          </Alert>
        </Col>
        {this.Redirect_sent()}
        {this.Error()}
      </div>
    )
  }
}
```

A.3 Componente BloodPresure

```
axios.defaults.xsrfCookieName = 'csrftoken'
axios.defaults.xsrfHeaderName = 'X-CSRFToken'
export default class BloodPresure extends React.Component {
    state = {
        value: '',
        activate_event: false,
        error_event: false,
    }
    componentWillMount() {
    this.props.location.result.service.then(service => {
     return service.getCharacteristic(this.props.location.result.
     characteristic)
   })
   .then(characteristic => {
      return characteristic.readValue();
   })
   .then(value => {
      this.setState({value: value.getUint8(0)})
   })
    .catch(error => {
      this.setState({error_event: true})
    });
  }
```

```
Repeat = () => {
  this.props.location.result.pressuredevice.gatt.connect()
  .then(server => {
    return server.getPrimaryService('blood_pressure')})
  .then(service => {
    return service.getCharacteristic(this.props.location.result.
    characteristic)
  })
  .then(characteristic => {
    return characteristic.readValue();
  })
  .then(value => {
    this.setState({value: value.getUint8(0)})
  })
  .catch(error => {
    this.setState({error_event: true})
  });
}
Send = () => {
  const blood_presure = {
    value: this.state.value,
    patient: this.props.location.result.id_patient,
    idnurse: this.props.location.result.id_nurse
  };
  axios.post('/save_blood_presure/', blood_presure)
  .then(res => {
    if(res.data=="ok"){
```

```
        this.setState({activate_event:true})
      }
       else{
        this.setState({error_event: true})
      }
    })
  }
  Redirect_sent = () =>{
    if(this.state.activate_event){
      return <Redirect to={{pathname: '/sent/',state: {id:
      this.props.location.result.id_nurse}}}/>
    }
  }
  Error = () =>{
    if(this.state.error_event){
      return <Redirect to={{pathname: '/error/',state:
      {id: this.state.idn}}}/>
    }
  }
  render() {
    return (
      <div className="BloodPressure">
        <Col xs={6} sm={6} md={6} lg={6}>
          <Alert variant="danger" style={{
              width: 400,
              height: 170,
              borderRadius: 10,
```

```
              marginTop: 25
            }}>
            <Alert.Heading>Resultado medición</Alert.Heading>
              <h5>{this.state.value}</h5>
            <hr />
            <div className="d-flex justify-content-end">
              <Col xs={9} sm={9} md={9} lg={9}>
                <Button variant="outline-danger" onClick={this.Repeat}>
                  Repetir medición
                </Button>
              </Col>
              <Col xs={3} sm={3} md={3} lg={3}>
                <Button variant="outline-danger" onClick={this.Send}>
                  Enviar
                </Button>
              </Col>
            </div>
          </Alert>
        </Col>
        {this.Redirect_sent()}
        {this.Error()}
      </div>
    )
  }
}
```

A.4 Componente HeartRate

```
axios.defaults.xsrfCookieName = 'csrftoken'
axios.defaults.xsrfHeaderName = 'X-CSRFToken'
export default class HeartRate extends React.Component {
    state = {
        value: '',
        error_event: false
   }
  componentWillMount() {
    this.props.location.result.service.then(service => {
     return service.getCharacteristic(this.props.location.result.
     characteristic)
   })
   .then(characteristic => {
      return characteristic.readValue();
   })
   .then(value => {
      this.setState({value: value.getUint8(0)})
   })
    .catch(error => {
        console.log('Argh! ' + error);
    })
  }
  Repeat = () => {
    this.props.location.result.heartdevice.gatt.connect()
    .then(server => {
```

```
      return server.getPrimaryService('heart_rate')})
    .then(service => {
      return service.getCharacteristic(this.props.location.result.
      characteristic)
    })
    .then(characteristic => {
      return characteristic.readValue();
    })
    .then(value => {
      this.setState({value: value.getUint8(0)})
    })
    .catch(error => {
      console.log('Argh! ' + error);
    });
  }
  Send = () => {
    const heart_rate = {
      value: this.state.value,
      patient: this.props.location.result.id_patient,
      idnurse: this.props.location.result.id_nurse
    };
    axios.post('/save_heart_rate/', heart_rate)
    .then(res => {
      if(res.data=="ok"){
        console.log(res.data)
        this.setState({activate_event:true})
      }
```

```
      else{
        this.setState({error_event: true})
      }
    })
  }
  Redirect_sent = () =>{
    if(this.state.activate_event){
      return <Redirect to={{pathname: '/sent/',state: {id:
      this.props.location.result.id_nurse}}}/>
    }
  }
  Error = () =>{
    if(this.state.error_event){
      return <Redirect to={{pathname: '/error/',state:
      {id: this.state.idn}}}/>
    }
  }
  render() {
    return (
      <div className="HeartRate">
        <Col xs={6} sm={6} md={6} lg={6}>
          <Alert variant="warning"
          style={{
              width: 400,
              height: 170,
              borderRadius: 10,
              marginTop: 25
```

```
            }}>
            <Alert.Heading>Resultado medición</Alert.Heading>
              <h5>{this.state.value} mm Hg</h5>
            <hr />
            <div className="d-flex justify-content-end">
              <Col xs={9} sm={9} md={9} lg={9}>
                <Button variant="outline-warning" onClick={this.Repeat}>
                  Repetir medición
                </Button>
              </Col>
              <Col xs={3} sm={3} md={3} lg={3}>
                <Button variant="outline-warning" onClick={this.Send}>
                  Enviar
                </Button>
              </Col>
            </div>
          </Alert>
        </Col>
        {this.Redirect_sent()}
        {this.Error()}
      </div>
    )
  }
}
```

A.5 Modelo

```
from django.db import models
from django.utils import timezone
import datetime
class Patient(models.Model):
    name = models.CharField(max_length=200)
    surname = models.CharField(max_length=200)
    id_patient = models.CharField(max_length=20, primary_key=True,
    default='SOME STRING')
    date_of_birth=models.DateField(default=datetime.date.today)
    def __str__(self):
        return self.name
    def __str__(self):
        return self.surname
    def __str__(self):
        return self.id_patient
class Temperature(models.Model):
    unique_together = (('id_temperature', 'patient'),)
    id_temperature = models.AutoField(primary_key=True)
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
    default='SOMESTRING')
    date=models.DateField(default=datetime.date.today)
    value = models.FloatField(null=True, blank=True, default=None)
class Alert(models.Model):
    unique_together = (('id_alert', 'id_patient'),)
    id_patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
```

```
    default='SOMESTRING')
    description = models.TextField(default='SOMESTRING')
    date=models.DateField(default=datetime.date.today)
    id_alert= models.AutoField(primary_key=True)
    urgent = models.BooleanField(default=True)
class BloodPressure(models.Model):
    unique_together = (('id_blood_pressure', 'id_patient'),)
    id_patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
    default='SOMESTRING')
    id_blood_pressure = models.AutoField(primary_key=True)
    date=models.DateField(default=datetime.date.today)
    value = models.FloatField(null=True, blank=True, default=None)
class HeartRate(models.Model):
    unique_together = (('id_heart_rate', 'id_patient'),)
    id_patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
    default='SOMESTRING')
    id_heart_rate = models.AutoField(primary_key=True)
    date=models.DateField(default=datetime.date.today)
    value = models.FloatField(null=True, blank=True, default=None)
class Doctor(models.Model):
    unique_together = (( 'dni','id'),)
    name = models.CharField(max_length=200)
    surname = models.CharField(max_length=200)
    id = models.CharField(max_length=20, primary_key=True,
    default='SOME STRING')
    dni = models.CharField(max_length=20, default='SOME STRING')
    date_of_birth=models.DateField(default=datetime.date.today)
```

```
    def __str__(self):
        return self.name
    def __str__(self):
        return self.surname
    def __str__(self):
        return self.id
class Doctor_account(models.Model):
    unique_together = (('user', 'id_doctor'),)
    user = models.CharField(max_length=20, primary_key=True,
    default='SOME STRING')
    password = models.CharField(max_length=20, default='SOME STRING')
    id_doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
    default='SOMESTRING')
class Nurse(models.Model):
    unique_together = (('dni','id'),)
    name = models.CharField(max_length=200)
    surname = models.CharField(max_length=200)
    id = models.CharField(max_length=20, primary_key=True,
    default='SOME STRING')
    dni = models.CharField(max_length=20, default='SOME STRING')
    date_of_birth=models.DateField(default=datetime.date.today)
    def __str__(self):
        return self.id
class Nurse_account(models.Model):
    unique_together = (('user', 'nurse_id'),)
    user = models.CharField(max_length=20, primary_key=True,
    default='SOME STRING')
```

```
    password = models.CharField(max_length=20, default='SOME STRING')
    nurse_id = models.ForeignKey(Nurse, on_delete=models.CASCADE,
    default='SOMESTRING')
class Get_temperature(models.Model):
    nurse=models.ForeignKey(Nurse, on_delete=models.CASCADE,
    default='SOME STRING')
    temperature=models.ForeignKey(Temperature, on_delete=models.CASCADE)
class Get_alert(models.Model):
    nurse=models.ForeignKey(Nurse, on_delete=models.CASCADE,
    default='SOME STRING')
    alert=models.ForeignKey(Alert, on_delete=models.CASCADE)
class Get_blood_pressure(models.Model):
    nurse=models.ForeignKey(Nurse, on_delete=models.CASCADE,
    default='SOME STRING')
    blood_pressure=models.ForeignKey(BloodPressure,
    on_delete=models.CASCADE)
class Get_heart_rate(models.Model):
    nurse=models.ForeignKey(Nurse, on_delete=models.CASCADE,
    default='SOME STRING')
    heart_rate=models.ForeignKey(HeartRate, on_delete=models.CASCADE)
class Treated(models.Model):
    patient = models.ForeignKey(Patient, on_delete=models.CASCADE,
    default='SOMESTRING')
    nurse = models.ForeignKey(Nurse, on_delete=models.CASCADE,
    default='SOMESTRING')
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
    default='SOMESTRING')
```

A.6 Vista

```
def Check_User(request):
    dicc = {}
    received_json_data=json.loads(request.body)
    user= received_json_data['user']
    password = received_json_data['password']
    users_nurse=Nurse_account.objects.values('user')
    users_doctor=Doctor_account.objects.values('user')
    users_nurse = [''.join(list(u.values())) for u in users_nurse]
    users_doctor = [''.join(list(u.values())) for u in users_doctor]
    if user in users_nurse:
        u=Nurse_account.objects.get(user=user)
        if password==u.password:
            nurse_id=str(u.nurse_id)
            return JsonResponse({'response':'ok','id':nurse_id,
            'user':'nurse'})
        else:
            return HttpResponse("error")
    elif user in users_doctor:
        u=Doctor_account.objects.get(user=user)
        if password==u.password:
            doctor_id= str(u.id_doctor)
            return JsonResponse({'response':'ok','id':doctor_id,
            'user':'doctor'})
```

```
        else:
            return HttpResponse("error")
    else:
        return HttpResponse("error")
def Login(request):
    return render(request, 'index.html')
def Login_Pac(request):
    return render(request, 'index.html')
def Check_Pac(request):
    received_json_data=json.loads(request.body)
    id_pat=received_json_data['id_pat']
    try:
        Patient.objects.get(id_patient=id_pat)
        return HttpResponse("ok")
    except Exception as e:
        return HttpResponse(e)
def Measures(request):
    return render(request, 'index.html')
def Planner(request):
    return render(request, 'index.html')
def FormAlert(request):
    return render(request, 'index.html')
def Error(request):
    return render(request, 'index.html')
def SaveAlert(request):
    try:
        received_json_data=json.loads(request.body)
```

```
        patient= received_json_data['patient']
        alert = received_json_data['alert']
        id_nurse= received_json_data['id_nurse']
        u= received_json_data['u']
        pat=Patient.objects.get(id_patient=patient)
        alarm= Alert()
        alarm = Alert(id_patient=pat, urgent=u, description=alert)
        alarm.save()
        nurse= Nurse()
        nurse.id=id_nurse
        get_alert = Get_alert()
        get_alert= Get_alert(nurse=nurse, alert=alarm)
        get_alert.save()
        return HttpResponse("ok")
    except Exception as e:
        return HttpResponse(e)
def SentAlert(request):
    return render(request, 'index.html')
def Main_Nurse(request):
    return render(request, 'index.html')
def NurseHistorial(request):
    return render(request, 'index.html')
def MedicineHistorial(request):
    return render(request, 'index.html')
def NewPassword(request):
    return render(request, 'index.html')
def SentNewPassword(request):
```

```
    return render(request, 'index.html')
def ObtainTemperature(request):
    return render(request, 'index.html')
def ObtainBloodPressure(request):
    return render(request, 'index.html')
def ObtainHeartRate(request):
    return render(request, 'index.html')
def Patients(request):
    id=request.GET.get('id')
    user= request.GET.get('user')
    patients=[]
    alarms=[]
    id_patients=[]
    if user=="doctor":
        treated= Treated.objects.filter(doctor=id)
        for idpat in treated:
            patient= Patient.objects.filter(id_patient=idpat.patient)
            for pat in patient:
                alerts= Alert.objects.filter(id_patient=pat.id_patient)
                if alerts:
                    alarms.append("true")
                else:
                    alarms.append("false")
                patients.append(pat.name+ ' ' + pat.surname)
                id_patients.append(pat.id_patient)
    else:
        treated= Treated.objects.filter(nurse=id)
```

```
        for idpat in treated:
            patient= Patient.objects.filter(id_patient=idpat.patient)
            for pat in patient:
                alerts= Alert.objects.filter(id_patient=pat.id_patient)
                if alerts:
                    alarms.append("true")
                else:
                    alarms.append("false")
                    patients.append(pat.name+ ' ' + pat.surname)
                    id_patients.append(pat.id_patient)
    return JsonResponse({'patients':patients, 'alerts':alarms,
    'ids':id_patients})
def patients_list(request):
    users=Patient.objects.values('name')
    list_users = [''.join(list(pac.values())) for pac in users]
    return JsonResponse(list_users,safe=False)
def List_Patients(request):
    return render(request, 'index.html')
def List_Temperatures(request):
    values = []
    ps = []
    date = []
    dicc = {}
    patient=json.loads(request.GET.get('name'))
    id=patient['patient']
    pat=Patient.objects.get(id_patient=id)
    temperatures=Temperature.objects.filter(patient__id_patient=pat)
```

```
    for t in temperatures:
        values.append(t.value)
        date.append(t.date)
        get_temperature= Get_temperature.objects.filter
        (temperature=t.id_temperature)
        for g in get_temperature:
            nurse= Nurse.objects.filter(id=g.nurse)
            for n in nurse:
                ps.append(n.name + " " + n.surname)
    return JsonResponse({'value':values, 'ps': ps, 'date': date})
def List_Alert(request):
    description= []
    date = []
    ps = []
    id=[]
    patient=json.loads(request.GET.get('name'))
    idpat= patient['patient']
    pat=Patient.objects.get(id_patient=idpat)
    alerts= Alert.objects.filter(id_patient__id_patient=pat)
    for a in alerts:
        description.append(a.description)
        date.append(a.date)
        id.append(a.id_alert)
        get_alert= Get_alert.objects.filter(alert=a.id_alert)
        for g in get_alert:
            nurse= Nurse.objects.filter(id=g.nurse)
            for n in nurse:
```

```
                ps.append(n.name + " " + n.surname)
    return JsonResponse({'description':description, 'date': date,
    'nurse': ps, 'id':id})
def Delete_Alert(request):
    try:
        received_json_data=json.loads(request.body)
        instance = Alert.objects.get(
        id_alert=received_json_data['id'])
        instance.delete()
        return HttpResponse("ok")
    except Exception as e:
        return HttpResponse(e)
def Save_temperature(request):
    try:
        received_json_data=json.loads(request.body)
        value= received_json_data['value']
        patient = received_json_data['patient']
        id_nurse = received_json_data['idnurse']
        patient=Patient.objects.get(id_patient=patient)
        temperature= Temperature()
        temperature= Temperature(patient=patient,value=value)
        temperature.save()
        nurse= Nurse()
        nurse.id=id_nurse
        get_temperature = Get_temperature()
        get_temperature= Get_temperature(nurse=nurse,
        temperature=temperature)
```

```
        get_temperature.save()
        return HttpResponse("ok")
    except Exception as e:
        return HttpResponse(e)
def Save_blood_presure(request):
    try:
        received_json_data=json.loads(request.body)
        value= received_json_data['value']
        patient = received_json_data['patient']
        id_nurse = received_json_data['idnurse']
        patient=Patient.objects.get(id_patient=patient)
        blood_presure=BloodPressure()
        blood_presure=BloodPressure(id_patient=patient, value=value)
        blood_presure.save()
        nurse= Nurse()
        nurse.id=id_nurse
        get_blood_pressure=Get_blood_pressure()
        get_blood_pressure=Get_blood_pressure
        (nurse=nurse, blood_pressure=blood_presure)
        get_blood_pressure.save()
        return HttpResponse("ok")
    except Exception as e:
        return HttpResponse(e)
def Save_heart_rate(request):
    try:
        received_json_data=json.loads(request.body)
        value= received_json_data['value']
```

```
        pat = received_json_data['patient']
        id_nurse = received_json_data['idnurse']
        patient=Patient.objects.get(id_patient=pat)
        heartrate=HeartRate()
        heartrate=HeartRate(id_patient=patient, value=value)
        heartrate.save()
        nurse= Nurse()
        nurse.id=id_nurse
        get_heart_rate=Get_heart_rate()
        get_heart_rate=Get_heart_rate(nurse=nurse,
        heart_rate=heartrate)
        get_heart_rate.save()
        return HttpResponse("ok")
    except Exception as e:
        print(e)
        return HttpResponse(e)
def List_heart_rate(request):
    value = []
    ps = []
    date = []
    dicc = {}
    patient=json.loads(request.GET.get('name'))
    id= patient['patient']
    patient=Patient.objects.get(id_patient=id)
    heart_rates= HeartRate.objects.filter
    (id_patient__id_patient=patient)
    for r in heart_rates:
```

```
        value.append(r.value)
        date.append(r.date)
        get_heart_rate= Get_heart_rate.objects.filter
        (heart_rate=r.id_heart_rate)
        for g in get_heart_rate:
            nurse= Nurse.objects.filter(id=g.nurse)
            for n in nurse:
                ps.append(n.name + " " + n.surname)
    print(value)
    print(ps)
    print(date)
    return JsonResponse({'value':value, 'ps': ps, 'date': date})
def List_blood_presure(request):
    value = []
    ps = []
    date = []
    dicc = {}
    patient=json.loads(request.GET.get('name'))
    id= patient['patient']
    patient=Patient.objects.get(id_patient=id)
    blood_presure= BloodPressure.objects.filter
    (id_patient__id_patient=patient)
    for b in blood_presure:
        value.append(b.value)
        date.append(b.date)
        get_blood_pressure= Get_blood_pressure.objects.filter
        (blood_pressure=b.id_blood_pressure)
```

```
        for g in get_blood_pressure:
            nurse= Nurse.objects.filter(id=g.nurse)
            for n in nurse:
                ps.append(n.name + " " + n.surname)
    return JsonResponse({'value':value, 'ps': ps, 'date': date})
def ChangePassword(request):
    received_json_data=json.loads(request.body)
    user= received_json_data['user']
    old_password = received_json_data['old_password']
    new_password = received_json_data['new_password']
    typeuser =  received_json_data['usertype']
    if typeuser=="nurse":
        users_nurse=Nurse_account.objects.values('user')
        users_nurse = [''.join(list(u.values())) for u
        in users_nurse]
        if user in users_nurse:
            u=Nurse_account.objects.get(user=user)
            if old_password==u.password:
                try:
                    nurse=Nurse.objects.get(id=u.nurse_id)
                    nurse_account=Nurse_account()
                    nurse_account=Nurse_account(user=user,
                    password=new_password,nurse_id=nurse)
                    nurse_account.save()
                    return HttpResponse("ok")
                except Exception as e:
                    return HttpResponse(e)
```

```
            else:
                return HttpResponse("error")
        else:
            return HttpResponse("error")
    elif typeuser=="doctor":
        users_doctor=Doctor_account.objects.values('user')
        users_doctor = [''.join(list(u.values())) for u
        in users_doctor]
        if user in users_doctor:
            u=Doctor_account.objects.get(user=user)
            if old_password==u.password:
                try:
                    doctor=Doctor.objects.get(id=u.id_doctor)
                    doctor_account=Doctor_account()
                    doctor_account=Doctor_account(user=user,
                    password=new_password,id_doctor=doctor)
                    doctor_account.save()
                    return HttpResponse("ok")
                except Exception as e:
                    return HttpResponse(e)
            else:
                return HttpResponse("error")
        else:
            return HttpResponse("error")
```

Appendix B

Manual de instalación

B.1 Requisitos previos

1. Tener instalado python
2. Tener instalado Django
3. Disponibilidad de un navegador chrome o opera con versiones iguales y superiores a 56 0 43 respectivamente.

B.2 Conocimientos previos

- Manejo de la terminal de linux
- Conocimientos básicos del ámbito web

B.3 Ejecución del proyecto

Para la ejecución del proyecto, al disponer de dos servidores(servidor de node.js y servidor de django), uno donde se encuentra nuestra API REST y otro donde nuestro cliente se renderiza, debemos ejecutar dos servidores.

A continuación describiremos como ejecutar cada uno de ellos:

- **Servidor de node.js** Para la ejecución del servidor de node se deben seguir los siguientes pasos:
    1. Entrar en la carpera principal del proyecto
    2. Entrar en la carpera mysite
    3. Entrar en la carpera react

4. Ejecutar el comando:
   $ yarn start

- **Servidor de django** Para ejecutar el servidor de django debemos seguir los siguientes pasos:

  1. Entrar en la carpera del proyecto.
  2. Iniciar el entorno virtual mediante el siguiente comando:
     $ source myvenv/bin/activate
  3. Entrar en la carpeta mysite
  4. Ejecutar el servidor mediante el comando:
     $ python manage.py runserver

Una vez ejecutados ambos servidores el usuario podrá acceder a la aplicación abriendo el navegador y buscando la url:

- http://127.0.0.1:8000/login/

Esta url llevará al usuario a la pantalla principal de la aplicación, donde mediante la aportación de usuario y contraseña correctos podrá acceder a las funcionalidades de la aplicación.

Si el usuario desarrollador quisiese realizar cambios a nivel de base de datos podrá hacerlo mediante la interfaz de admin de django ejecutando en el navegador:

- http://127.0.0.1:8000/admin/

Esta url le llevará a un formulario donde se le será requerido un usuario o contraseña. Para obtener estos deberá crear un superusuario mediante el siguiente comando:

$ python manage.py createsuperuser

Para ejecutar el comando anterior correctamente debe asegurarse de estar posicionado en el path: \[carpeta del proyecto]\[carpeta mysite].

B.4 Aspectos a considerar

1. El usuario debe esperar a que ambos servidores se hayan ejecutado correctamente antes de iniciar la llamada http en el navegador.

2. El servidor de node.js abrirá de forma automática una ventana en el navegador cuando este se ejecute. Esta ventana podrá cerrarse una vez cargada esta.

3. Esta aplicación está implementada en local. Si se quisiese incluir esta en un sitio web, se deberá configurar un servidor https, pues de lo contrario no funcionará correctamente.

4. Si el usuario utiliza periféricos distintos a los proporcionados por la aplicación: LighBlue o utiliza esta no utilizando el nombre del device que se crea de forma automática cuando se selecciona el servicio, el usuario deberá cambiar la forma de identificación del device del componente bluetooth, estableciendo el device.name de los condiconales al correspondiente nombre que tenga el dispositivo medidor que se esté utilizando.

5. La aplicación LighBlue sólo permite crear dispositivos periféricos virtuales en iphone o ipad.